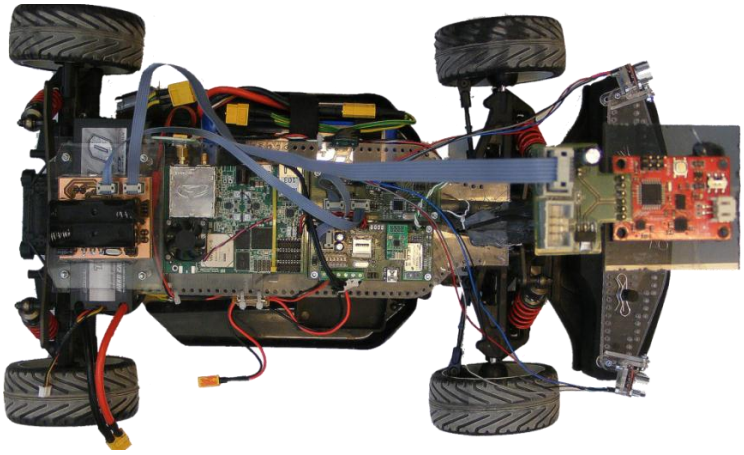
CHALMERS



Gulliver

– en plattform för testning, utveckling och demonstration av transportsystem

Ett kandidatarbete inom Data- och informationsteknik

Erik Dahlgren
Johan Grundén
Daniel Gunnarson
Nadia Holtryd
Anmar Khazal
Viktor Swantesson

Institutionen för Data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2012

Kandidatarbete/rapport nr 2012:061

Abstract

The bachelor project has created a test and development platform that facilitates the search for traffic solutions to current and future traffic problems. These problems include the increase of congestion and traffic accidents, as well as carbon dioxide emissions from all vehicles. The project's aim is to contribute to finding smarter and cheaper ways to solve such traffic related problems.

The project has integrated specialized subsystems such as a traffic simulator, a radio node simulator, real miniature vehicles and a map client to create an extensive test and development platform. The platform can create and simulate realistic traffic scenarios, and therefore provides an opportunity to try different solutions and improvements for today's traffic. Through the integration of physical miniature vehicles and simulated vehicles, it actualizes and simplifies testing of genuine and complex traffic phenomena. In addition to the testing of large scale traffic simulations and potential traffic solutions, the platform offers visual real-time updates of the vehicle's position as well as tools to create custom made scenarios. The platform has been verified through two test cases that implicates that the following functionality has been achieved; (a) a map client has been built, in which maps can be created and edited, to be used for testing, (b) a two-way real time communication between a traffic simulator and physical miniature vehicles has been implemented, (c) two-way communication between a traffic simulator and a node simulator that allows for simulation of multiple abilities, e.g. radio communication and decision making, between the miniature vehicles. A user can in real-time simulate traffic situations where a map has been designed and a mix of virtual and physical vehicles has been chosen.

Looking ahead, the developed system can be seen as the first stepping stone towards the development and realization of more secure and efficient transportation system.

Sammanfattning

Kandidatprojektet har tagit fram en test- och utvecklingsplattform som underlättar testning och förståelse av komplexa trafiksituationer. En förståelse som kan användas till att lösa så väl dagens som morgondagens trafikproblem. Dessa problem innefattar ökade köbildningar, en stigande olycksfrekvens samt den miljöpåverkan som trafikens koldioxidsutsläpp orsakar. Ett långsiktigt mål för detta projekt är att möjliggöra ett intelligentare och billigare sätt att lösa dessa problem.

Specialiserade delsystem som innefattar en trafiksimulator, en kommunikationsnod-simulator, fysiska miniatyrfordon samt en kartklient vilka har integrerats för att skapa en omfattande test- och utvecklingsplattform. Plattformen har kapacitet att återge verklighetstrogna trafiksituationer och ger därför en möjlighet att testa olika lösningar. Genom integrering av fysiska miniatyrfordon med simulerade fordon, realiseras och förenklas testning av komplexa trafikfenomen. Förutom testning av storskaliga trafiksimuleringar och potentiella trafiklösningar så erbjuder plattformen visuell realtidsuppdatering av fordonens position samt möjligheten att skapa egna skräddarsydda testfall och kartor. Plattformen har verifierats i två testfall som visar på att följande funktionalitet har uppnåtts; (a) en kartklient där kartor kan skapas och redigeras för att sedan användas i testfall har implementerats, (b) en tvåvägskommunikation i realtid mellan en trafiksimulator och fysiska miniatyrfordon har realiserats, (c) en tvåvägskommunikation mellan trafiksimulator och ett nodsimulatorprogram som kan simulera flera egenskaper, såsom radiokommunikation och fordon och beslutsfattande, mellan de fysiska miniatyrfordonen. Med hjälp av systemet kan en användare i realtid simulera trafiksituationer där en karta av ett scenario designats och en blandning av virtuella och fysiska fordon valts.

Den utvecklade plattformen är ett steg framåt mot ett billigare, säkrare och effektivare sätt att testa framtidens trafiklösningar.

Förord

Rapporten är ett kandidatarbete vid Institutionen för Data- och informationsteknik på Chalmers Tekniska Högskola och utfördes våren 2012. Projektgruppen önskar tacka handledare Elad Schiller för hans stora engagemang samt för all den tid han har lagt ned på att rådgiva och stötta oss genom projektet. Projektgruppen vill även tacka Benjamin Vedder, Mohamed Mustafa, Mitra Pahlavan och Chalmers Robotförening för all hjälp och rådgivning under projektets gång. Ett stort tack riktas även till projektets sammarbetspartners i kandidatgruppen DATX01, bestående av Alexander Altby, Tobias Boström, Timur Sibgatullin och Karl Stjärne, som stod för positioneringssystemet till de fysiska robotarna. Vi vill också tacka Andreas Larsson för hans åsikter och tankar kring denna rapport. Slutligen vill vi även tacka SAFER Fordons- och trafiksäkerhetscentrum på Chalmers.

Ordlista

Drag and drop: (Drag och släppa) Begrepp inom grafiska användargranssnitt där användaren greppar tag i virtuella objekt och sedan förflytta dem till en annan plats.

Exekvera: Innebär detsamma som att utföra, genomföra eller verkställa. I utvecklingssammanhang används det ofta i kontexten "exekvera kod", det vill säga "att köra kod".

GUI: Graphical User Interface, hänvisar till det visuella gränssnitt som möjliggör interaktion mellan användare och datorsystem.

Internminne: Minne där data och program som exekveras lagras för direkt åtkomst av processorn under körning.

Nodsimulator: Simulator som används för att testa sensornodsprogram virtuellt i en PC och simulera en radiomiljö.

Open source: Filosofi i utvecklingssammanhang som innebär att källkoden eller ritningarna bakom ett system skall vara öppna för alla att granska, använda och ändra.

Parprogrammering: Utvecklingsmetod där två utvecklare sitter vid samma dator och hjälps åt att utveckla program. Leder till högre kvalitet på koden samt underlättar problemlösning av komplexa problem (Cockburn, Williams, 2000).

Platooning: Strategi för att placera fordon i täta kolonner på vägar. Kolonnen leds ofta av en yrkeschaufför som kör manuellt, exempelvis en lastbilschaufför. Fordonen bakom körs autonomt med hjälp av datorstyrt beslutsfattande och sensorer. Metoden minskar både utsläpp och ökar trafikgenomströmningen.

Processor: Krets i ett digitalt system som utför någon typ av beräkning.

Programmeringskort: Fysiskt kort som agerar brygga mellan en PC och en programmerbar enhet, ofta någon typ av inbyggt system. Möjliggör programmering och konfigurering av den externa enheten.

Qt: Qt [kju:t] är ett applikationsramverk som lämpar sig för utveckling i framförallt C++ och har många grafiska finesser som är enkla att använda.

Sensor: Begrepp för en apparat eller anordning som har till uppgift att känna av värdet på någon fysisk storhet, exempelvis tryck eller temperatur. (Nationalencyklopedin, 2012)

Seriell expansionsport: Kommunikationsport som möjliggör seriell kommunikation med något utbytbart tillbehör, exempelvis en sensor.

TCP: Transmission Control Protocol, är ett dataöverföringsprotokoll som tillhandahåller en dataström mellan två datorer eller en koppling mellan olika program på en dator.

Trafiksimulator: Verktyg för att simulera olika former av trafikmiljöer. Simulatorn har även implementerade algoritmer för mänskligt förarbeteende.

Tråd: Minsta nedbrytbara exekvering av ett programavsnitt som kan schemaläggas och hanteras av ett operativsystem. Om flera olika exekveringar måste utföras samtidigt kan de placeras i olika trådar och spridas ut över flera processorkärnor för att tillåta parallell exekvering (Sutter, 2005).

WiFi: Wireless Fidelity, är en populär standard för trådlöst nätverk som är vanligt förekommande främst i Pc-datorer. Möjliggör trådlös överföring av data i höga hastigheter.

1 INLEDNING 1

1.1 Bakgrund 1
1.2 Syfte 2
1.3 Målsättning 2
1.4 Avgränsningar 3
1.5 Utvecklingsmetodik 3
1.6 Versionshantering 3
1.7 Systemvalidering 4
1.8 Disposition 4

2 PROBLEMFORMULERING 5

2.1 Kartklient 5
2.1.1 Kartklientens egenskaper 5
2.1.2 Design av grafiskt användargränssnitt 5
2.1.3 Videoström från Gulliver-robot 5
2.2 Kommunikation mellan fysiska- och datorsimulerade virtuella fordon 6
2.2.1 Kartklient till trafiksimulator 6
2.2.2 Trafiksimulator till fysisk Gulliver-robot 6
2.2.3 Trafiksimulator till nodsimulator 6
2.2.4 Radiosimulering 6

3 CENTRALA TEKNIKER OCH VERKTYG 8

3.1 Gulliver-robot 8
3.2 Kommunikationsenheten MICAz 8
3.3 TinyOS 9
3.3.1 TinyOS-simulatorn TOSSIM 9
3.4 Trafiksimulatorn SUMO 10
3.4.1 Algoritmer för förarbeteende 12
3.4.2 Traffic Control Interface 13
3.5 Videoströmning 13
3.6 Generella algoritmer 13
3.6.1 Två linjers skärningspunkt 14
3.6.2 Grafisk algebra 14
3.7 Programmeringsspråk 15
3.7.1 XML 15
3.7.2 NesC 15
3.7.3 Python 15
3.7.4 Java 16
3.7.5 C++ 16

4 RESULTAT 17

4.1 Implementering av plattformen 17
4.1.1 Kartklienten 18

4.1.1.1 Utvecklingsverktyget Qt ... 18
4.1.1.2 Kartklientens egenskaper ... 18
4.1.1.3 Design av grafiskt användargränssnitt ... 21
4.1.1.4 Implementering av videoström ... 22
4.1.2 Kommunikation och integration ... 22
4.1.2.1 Trafiksimulatorn ... 22
4.1.2.2 SUMO till delsystemen ... 23
4.1.2.3 MICAz till Gulliver-robot ... 25
4.1.2.4 SUMO till TOSSIM ... 25
4.1.2.5 Kartklienten till SUMO och Gulliver-robot ... 26
4.1.3 Förarlogik hos Gulliver-robot ... 26
4.2 VALIDERING AV PLATTFORMEN ... 26
4.2.1 Testfall - Integration av virtuellt och fysiskt fordon ... 26
4.2.2 Testfall - TOSSIM-simulerade fordon ... 27
4.2.3 Funktionsvalidering ... 28

5 DISKUSSION OCH REFLEKTION ... 31

5.1 UTVECKLINGSPROCESS ... 31
5.2 PLATTFORMENS FUNKTIONALITET ... 32

6 SLUTSATSER ... 34

7 REFERENSER ... 35

BILAGA A - ANSVARSOMRÅDEN ... 37

A.1 ANSVARSOMRÅDEN ... 37
A.2 FRAMTAGNING AV SLUTRAPPORT ... 37
A.3 PRESENTATIONER OCH OPPONERING ... 37

BILAGA B - INSTALLATION AV TINYOS ... 38

BILAGA C - LÖSNING PÅ PROBLEM I SUMO ... 41

BILAGA D - VIDAREBEFORDRAD KUNSKAP ... 42

1 Inledning

I Europa är transportrelaterade olyckor den enskilt största orsaken till dödsfall i åldersgruppen 15 till 29 år (Eurostat, 2011). Enbart i Sverige skadades över 20 000 personer i trafiken år 2010 (Transportstyrelsen, 2010). Om antalet fordon i trafiken inte minskar avsevärt kommer vi att stå inför en mängd svåra problem de kommande åren (Myhr, 2010). Längre bilköer, fler trafikolyckor med medföljande stora sociala kostnader, ökat behov av vägnät och ökade koldioxidutsläpp är några exempel.

För att lösa existerande samt förutsedda framtida problem har många nya innovativa lösningar föreslagits. Ett exempel på en sådan lösning är SARTRE (SAfe Road TRains for the Environment) (Chan, Coelingh, Robinson, 2010), ett projekt som den Europeiska kommissionen finansierar. Projektet testar ett koncept kallat platooning, där flera bilar kör i ett tåg och endast ledarbilen styrs av en mänsklig förare. Målet för SARTRE är att integrera dessa tågsystem på motorvägar. Konceptet platooning effektiviserar trafikflödet på vägarna och är en möjlig lösning på exempelvis ett ökat behov av vägnät eller bilköer. Då det handlar om komplicerade och säkerhetskritiska system krävs noggrann och uttömmande testning för att verifiera lösningarna. Det är därför relevant att utveckla en plattform för att på ett prisvärt, effektivt och lättillgängligt sätt kunna testa trafiksituationer med olika lösningar. Genom att använda så kallade miniatyrfordon och virtuella fordon i en testmiljö minskar kraven på både utrymme och kostnad avsevärt. Dessa två faktorer påverkar tillgängligheten av systemet markant vilket i sin tur möjliggör för fler aktörer att etablera sig på marknaden för trafiklösningar. Fler aktörer leder till ökad konkurrens vilket även kan bidra till snabbare framsteg inom området.

Kandidatarbetets föreslagna test- och utvecklingsplattform är designad för att kombinera fysiska och virtuella fordon i en och samma testmiljö. På så sätt upplever de fysiska fordonen att de befinner sig i en trafiksituation med tusentals andra fordon som i själva verket är datorsimulerade. När fysiska och virtuella fordon integreras kallas detta ett cyberfysiskt system. Här kan beteenden som uppstår i komplexa trafiksituationer, där tusentals fordon är inblandade, observeras med hjälp av ett fåtal fysiska fordon. Då undviks behovet av att ha ett stort antal fullskaliga fordon och mänskliga förare. Eftersom de fysiska och virtuella fordonen behöver ingå i samma testmiljö och behöver ha en gemensam uppfattning om hur omgivningen ser ut behövs en gemensam karta. För att öka möjlighet till en större överblick och kontroll över testmiljön ingår det även ett gränssnitt för att skapa och redigera denna miljö. Testplattformen ökar således möjligheterna att på ett resurssnålt vis testa framtidens potentiella trafiklösningar.

1.1 Bakgrund

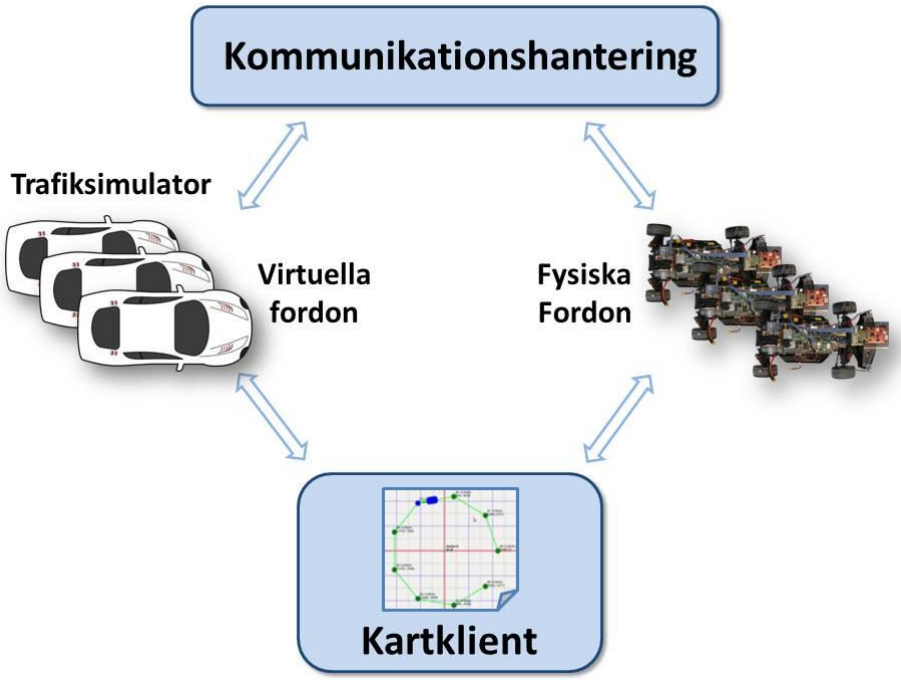
Kandidatarbetet har inspirerats av forskningsprojektet Gulliver som startades hösten 2011 på Chalmers Tekniska Högskola. Vid kandidatarbetets start innehöll forskningsprojektet en konceptidé för att tillhandahålla en smidig och billig testmiljö för forskning inom intelligenta transportsystem (Pahlavan, Papatriantafilou, Schiller, 2011). Det hade inom projektet skapats två robotar i samarbetet med Chalmers robotförening samt kod för styrning av dessa. Ett virtuellt-trafikljus var även under utveckling. Kandidatarbetet har använt dessa robotar, i rapporten kallade Gulliver-robotar, för att validera den plattform som utvecklats.

1.2 Syfte

Kandidatarbetet syftar till att skapa en test- och utvecklingsplattform för framtidens fordonssystem. Denna plattform är ämnad att testa möjliga lösningar på de trafikproblem som är beskrivna i inledningen. Slutprodukten integrerar fysiska miniatyrfordon med datorsimulerade virtuella fordon för att effektivisera och öka takten för utvecklingen av framtidens fordonsteknik. Rapporten ämnar dokumentera processen att utveckla en sådan plattform.

1.3 Målsättning

Målet för kandidatarbetet är att hitta lösningar för integrering av kommunikation mellan fysiska miniatyrfordon och datorsimulerade virtuella fordon, detta kräver att det skapas en kartklient, se figur 1.1. Kartklienten möjliggör en visuell realtidsuppdatering av fordonens position samt bidrar med verktyg för framtagandet av testfall. En integrering av fysiska miniatyrfordon och simulerade fordon realiserar samt förenklar testning av verkliga och komplexa trafikfenomen. Kommunikationen mellan plattformens delar bidrar till en bättre förståelse av interaktionen mellan fordonen. Genom att plattformen sammankopplar specialiserade delsystem som trafiksimulering, virtuell nodsimulering samt fysiska miniatyrfordon, kan verklighetstrogna resultat uppnås. Detta ger plattformen en högteknologisk slagkraft vilken gör den användbar för forskning inom framtidens fordonssystem.



Figur 1.1: Visar den plattform som projektet har som målsättning att skapa.

I kapitel 2 specificeras målet närmare genom uppdelning i delproblem för att skapa mer förståelse och översikt över systemet.

1.4 Avgränsningar

Kandidatarbetet har fokuserats på att implementera en prototypplattform som öppnar upp och förbereder för resurseffektiv testning samt framtagning av smarta trafiklösningar. Detta innebär att projektet inte har behandlat sådant som att bygga strategier för möjliga trafikförbättringar.

1.5 Utvecklingsmetodik

Arbetet har skett i två mindre grupper. Den ena gruppen, om fyra personer, arbetade med att sammankoppla plattformens olika delsystem och den andra gruppen, om två personer, utvecklade tjänsten för karthantering. Kandidatarbetet har använt en iterativ utvecklingsmetod. Eftersom lösningen för målet har vuxit fram under arbetets gång har det varit av hög prioritet att ha stor frihet i att ändra tekniker för att nå målet.

I de båda grupperna har två större utvecklingsomgångar genomförts. Omgångarna har följts av testning och utvärdering. En systemdesign har vuxit fram under projektets gång och implementeringen av denna design är även en validering av de designval som gjorts. Den första prototypen av plattformen inklusive kartklienten var klar då halva projekttiden hade gått och innehöll delar av det slutgiltiga systemet. Prototypen innehöll en förenklad trafiksituation för fordon, denna visade centrala delar av den funktionalitet som den slutgiltiga plattformen behövde. Den mindre gruppen skulle skapa enklare kartegenskaper för att möjliggöra olika behövda trafikmiljöer i kartklienten. Efter testning och utvärdering av den första prototypen, utvecklades och slutfördes det kompletta systemet.

Eftersom plattformen måste vara lättillgänglig och öppen för vidare användning beslutades det att endast använda *open source* baserade verktyg, d.v.s. program med öppen och fritt tillgänglig källkod.

1.6 Versionshantering

Projektet har utvecklats modulärt där projektmedlemmarna ensamma eller med hjälp av parprogrammering varit ansvariga för att utveckla respektive delsystem. Den färdiga plattformen är sammansatt av flera mindre delar som bygger på olika tekniker. Från början var respektive gruppmedlem ansvarig för att själv versionshantera sin kod lokalt i den egna utvecklingsmiljön. Under integrationsfasen utsågs en ansvarig person som hade ansvar för att versionshantera all kod med hjälp av versionshanteringssystemet Subversion (Apache, 2012).

För planering, möten och andra dokument användes Google-docs (Google, 2012), vilket tillåter att flera användare kan skriva och granska samma dokument samtidigt. Denna tjänst tillåter att alla gemensamma filer lagras på en plats vilket medfört en snabbare och effektivare arbetsgång genom hela projektet.

1.7 Systemvalidering

För att validera den plattform som projektet utvecklat har systemtester genomförts. Valideringen är nödvändig för att upptäcka fel och ofullständigheter i implementeringen. Testerna är en konstruktion där den önskade funktionaliteten prövades. Redogörelse för dessa tester finns i avsnitt 4.2.

1.8 Disposition

Rapporten börjar med att närmare specificera de problem som projektet har ställts inför för att skapa den önskade plattformen. Sedan redovisas de tekniker och verktyg som haft stor betydelse för arbetet. Dessa tekniker och verktyg ligger till grund för den plattform som skapades inom kandidatarbetet. I nästkommande kapitel beskrivs hur projektet har skapat den önskade plattformen och därefter redogörs för den validering, med systemtester som genomförts, för att bekräfta måluppfyllnad. I kapitlet därefter följer en diskussion som utvärderar arbetsmetodiken samt testsystemet. Avslutningsvis presenteras olika möjliga vidare utvecklingar och slutsatser kring det genomförda projektet.

2 Problemformulering

I föregående kapitel gavs en inledning och bakgrund till projektet samt dess mål. I detta kapitel beskrivs den problemställning som följer av att skapa det plattform som projektets mål kräver. Här beskrivs kartklienten och de egenskaper den behöver ha. Slutligen beskrivs de olika simulatorerna och kommunikationen mellan dem, som krävs för att skapa den virtuella miljö som fordonen använder.

2.1 Kartklient

För att uppnå de mål som ställts på kandidatarbetet krävs det en kartklient som kan styra och manövrera över Gulliver-robotarna. Denna klient låter användaren skapa kartor med specifik information, såsom hastighet och ID-nummer. Klienten får tillbaka realtidsinformation om var fordonen befinner sig så att detta kan förmedlas tillbaka till användaren. För att göra kartklienten tillgänglig för användare vars expertis inte nödvändigtvis är teknisk, krävs ett användarvänligt grafiskt gränssnitt. Detta gränssnitt bör således implementeras på en utvecklingsplattform som tillåter grafisk implementation.

2.1.1 Kartklientens egenskaper

I kartklienten måste det vara möjligt att konstruera sina egna vägnät. Vid själva skapandet av vägnäten krävs det verktyg som är enkla för användaren att använda för att skapa komplexa trafiksystem och miljöer. Verktyg som exempelvis gör det möjligt att lägga till och ta bort vägpunkter (så kallade noder), samt att rotera, förstora och förminska vägsegment. Användaren måste även ha möjligheten att spara och läsa in skapade vägnät till och från roboten samt att kunna verifiera kartans vägnät. Med att verifiera ett vägnät menas att användaren varnas där rutter korsar varandra, ligger för nära varandra eller om ett vägsegment har en för snäv kurva. Problematiken i detta moment är att de flesta av verktygen kräver avancerad matematik samt logik för att skapas, detta för att bibehålla den standard som klienten måste tillhandahålla.

2.1.2 Design av grafiskt användargränssnitt

Kartklienten är en central del av projektet som öppnar upp för enklare och utförligare testning av framställda teknologier inom området. Designmässigt sett finns det några viktiga punkter som bör framhållas:

- Ett intuitivt gränssnitt som gör att användaren snabbt blir produktiv med verktyget.
- Att användaren får en god upplevelse vid användandet av klienten.

2.1.3 Videoström från Gulliver-robot

Genom en realtidsvideoström ges användaren djupare insikt i hur Gulliver-robotarna beter sig i det aktuella testfallet. Strömmen skickas via en webbkamera som är monterad på robotens framsida där den mänskliga föraren skulle befunnit sig. Visualiseringen av bilden finns i kartklienten, på så vis får användaren en större överblick av dynamiken och trafiken i det uppsatta testfallet. Problemen som då kan uppkomma är att hitta ett lämpligt sätt att över WiFi eller annan kommunikationsförbindelse föra över video från Gulliver-roboten till kartklienten.

2.2 Kommunikation mellan fysiska- och datorsimulerade virtuella fordon

I slutprodukten måste det vara möjligt för samtliga komponenter i systemet att kommunicera med varandra för att skapa en sammanhållen plattform. Att dessa komponenter aldrig varit integrerade i ett och samma system innebär stora utmaningar. De måste exempelvis vara kompatibla med varandra och ha möjlighet att tolka information på exakt samma sätt. Då slutprodukten har det stora problemet att den måste stödja simuleringar där tusentals fordon, fysiska samt virtuella, kommunicerar och interagerar med varandra krävs en lösning för detta.

2.2.1 Kartklient till trafiksimulator

För att möjligöra en sammansatt plattform där både Gulliver-robotar och de simulerade fordonen samverkar, behövs en "gemensam verklighet" d.v.s. en gemensam karta. Kartan som robotarna och de simulerade fordonen kör enligt och får sin position ifrån måste vara densamma så att båda kan jämföra sin placering på ett korrekt sätt. Utmaningen ligger i att det måste finnas ett sätt att transformera informationen från kartan i kartklienten till kartan i trafiksimuleringsverktyget.

2.2.2 Trafiksimulator till fysisk Gulliver-robot

Eftersom den fysiska Gulliver-roboten är en del av simuleringen i simuleringsprogrammet och kan stöta på hinder i verkligheten som ej kan förutses i simuleringen, så behöver den skicka information till programmet om exempelvis dess uppdaterade position eller fart. Simuleringsprogrammet i sin tur behöver skicka information till robotarna om positionen för fordon som endast befinner sig i simuleringen. Svårigheten uppstår då i länkandet mellan simuleringsprogrammet och Gulliver-roboten där information skickas i realtid, i båda riktningar. Både de datorsimulerade och de fysiska robotarna behöver informationen som skickas, på ett identiskt sätt. De behöver t.ex. tolka informationen i samma storheter såsom att hastighet inte tolkas som km/h på den ena och m/s på den andra.

2.2.3 Trafiksimulator till nodsimulator

En trafiksimulator i kombination med fysiska Gulliver-robotar ger en kraftfull och verklighetsnära plattform för att testa transportsystem. Många lösningar för framtidens transportsystem innebär att fordon kan prata med varandra och trådlöst utbyta information och därefter assistera föraren på olika sätt. Denna möjlighet finns på roboten tack vare MICAz-noden men samma system och möjligheter finns inte implementerat i någon trafiksimulator. För att tillåta storskalig testning med tusentals fordon måste även de simulerade fordonen stödja samma typ av kommunikation mellan varandra. Därför måste en nodsimulator som kan exekvera samma kod som de fysiska fordonen implementeras med möjlighet till snarlika kommunikationsmöjligheter och samma möjligheter att fatta förarbeslut.

2.2.4 Radiosimulering

För att kunna emulera en liknande radiomiljö till de simulerade fordonen som den de fysiska fordonen har, måste en lämplig skiftande, simulerad radiomiljö skapas. Utmaningen ligger i implementeringen av denna dynamiska miljö. Eftersom att fordonen hela tiden rör på sig förändras avståndet mellan dem vilket har stor påverkan på radiosignalerna. Detta innebär att radiosignalerna hela tiden måste uppdateras och anpassas till fordonens olika positioner. Vid

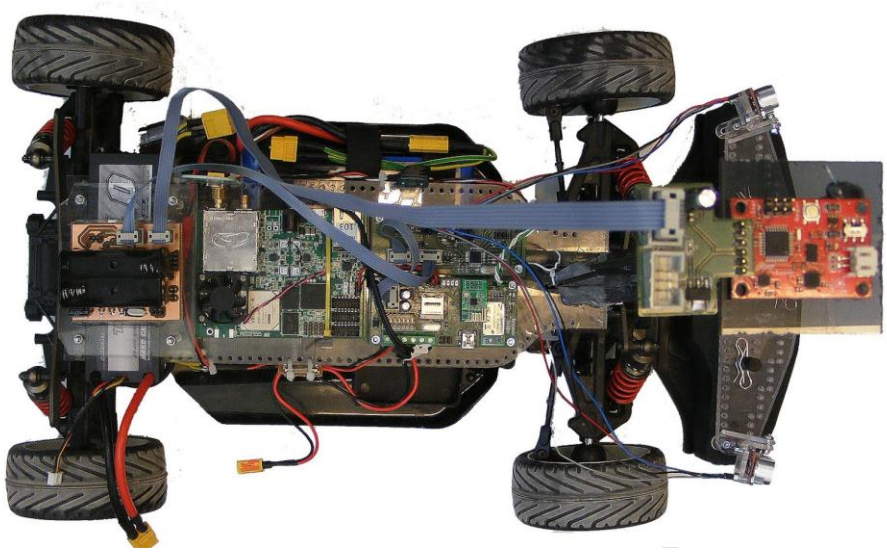
initiering av simulationer måste det också vara enkelt att tilldela fasta variabler som exempelvis styr verklighetstrogna bakgrundsstörningar för radio i den miljö som är önskvärd att testa i.

3 Centrala tekniker och verktyg

I föregående kapitel beskrivs de delproblem som implementeringen av testsystemet kräver. I detta kapitel beskrivs de tekniska begrepp och verktyg från den problemställningen som framställts i tidigare kapitel. Detta för att ge insikt om varför de specifika verktygen har inkluderats och använts i arbetet, samt för att få en grundläggande förståelse över vad dessa begrepp och verktyg innefattar.

3.1 Gulliver-robot

Gulliver-robotarna har en programmerbar *main board* som det går att ladda upp kartor till. Robotarna har även en specialdesignad motorkontroller som möjliggör styrning av hastigheten. De robotar som projektet använt sig av, se figur 3.1, har en PC som använder operativsystemet Linux för att möjliggöra bland annat videoströmning. Roboten har också ett nyutvecklat, högupplöst lokaliseringssystem som hjälper den att bestämma sin position. Denna funktion har utvecklats av kandidatgrupp DATX01 parallellt med utvecklingen av detta kandidatarbete (Altby et al, 2012).



Figur 3.1: Gulliver-roboten

3.2 Kommunikationsenheten MICAz

För att koppla trådlösa kommunikationsmöjligheter samt beslutsfattande logik till Gulliver-robotarna är en liten trådlös sensornod placerad på varje fordon. Gulliver-robotarna är utrustade med sensornoder av typen MICAz se figur 3.2 (Crossbow, 2008). En MICAz är en liten energisnål, trådlös sensornod med ett radiogränssnitt (802.15.4) (Xia, F. et. al. 2011), för kommunikation med andra liknande noder. Den har även en seriell expansionsport som möjliggör kommunikation med en PC. De är också utrustade med en enkel processor (8 MHz) och en liten mängd internminne. Sensornoderna använder operativsystemet TinyOS och kan exekvera program skrivna i programmeringsspråket NesC.

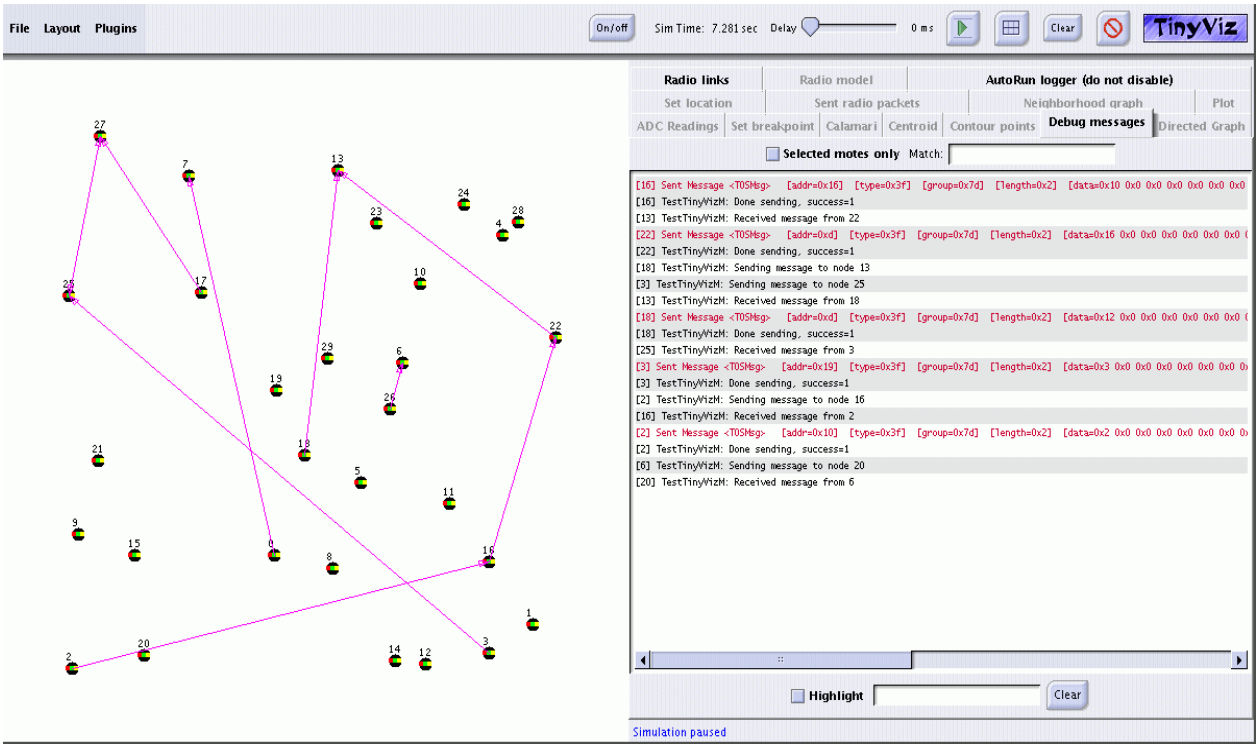
Figur 3.2: MICAz-noder

3.3 TinyOS

TinyOS (Levis, Gay, 2009) är ett operativsystem som är skapat för att köras på små inbyggda system med låg energiförbrukning som exempelvis trådlösa sensornoder som MICAz. Systemet tillhandahåller en komplett verktygslåda för att konstruera, kompilera och testa program för sensornoder. Program skrivs huvudsakligen i språket NesC, som är en variant av C med ett antal abstraktioner för att underlätta utveckling av program för sensornoder. Exempel på abstraktion som tillhandahålls är att det är enkelt att skicka radiomeddelanden mellan noder. Det är möjligt att direkt specificera en önskad meddelandestruktur utan att behöva tänka på bitströmmar eller andra lågnivåtekniker som är nödvändiga för kommunikation. Systemet är händelsebaserat vilket innebär att metoder definieras för olika händelser som inträffar. Ett exempel på detta kan vara om en timer aktiveras eller ett meddelande tas emot på radion. Då kommer en metod köras för att hantera den aktuella typen av händelse. Detta kan vara att läsa av ett sensorvärde eller göra någon beräkning eller åtgärd för att hantera det mottagna meddelandet.

3.3.1 TinyOS-simulatorn TOSSIM

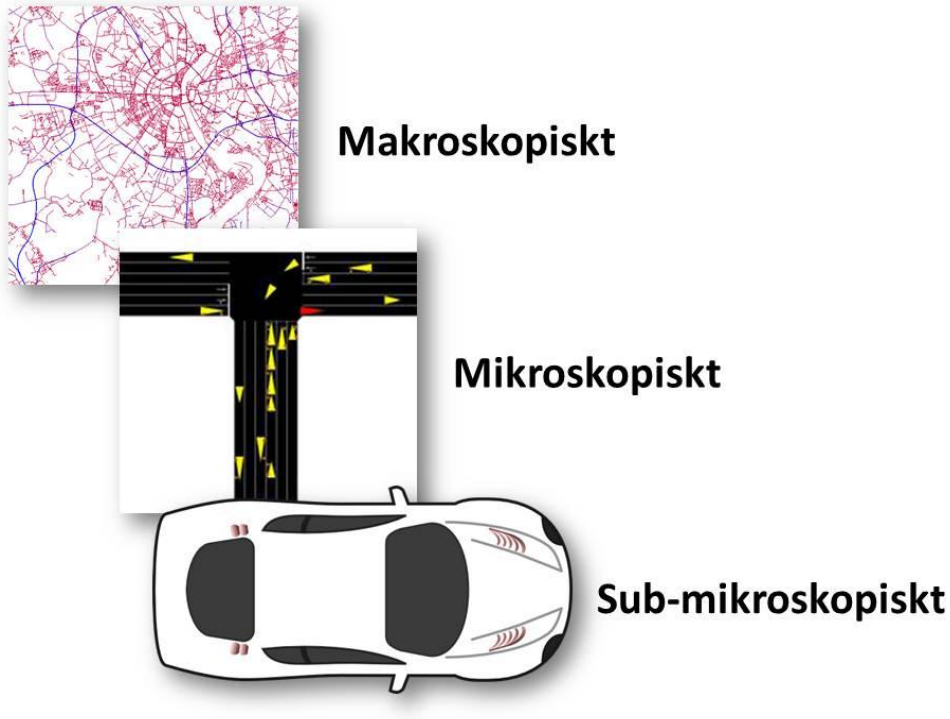
TOSSIM (TinyOS mote SIMulator), se figur 3.3, är en simulator för att testa sensornodsprogram virtuellt i en PC. TOSSIM gör det möjligt att testa och köra ett helt system med många noder virtuellt utan att behöva koppla in hårdvara (Culler D et al, 2003). Programmet skapar en virtuell miljö som stöder testning och felsökning av samma kod som körs i sensornoder, exempelvis MICAz, se avsnitt 3.2. TOSSIM ger också tillgång till en virtuell abstraktion av sensornodernas radio som kan emulera komplexa radiomodeller. Det finns bland annat möjlighet att simulera bakgrundsstörningar, paketkrockar, paketförluster samt simulerade avstånd mellan noder som påverkar radiomiljön (Levis, P. Rusak, T. 2010). TOSSIM har även möjlighet till kommunikation med "världen utanför" via en TCP-server som möjliggör att program utanför kan kommunicera med noder i simulationen i realtid. Simulatorn förenklar felsökning då det är möjligt att enkelt skapa loggfiler eller se händelserna i realtid i simuleringsmiljön, något som inte är möjligt om hårdvara används. TOSSIM kontrolleras antingen via Python eller C++, se avsnitt 3.7.3 och 3.7.5, som erbjuder stor frihet i sin konfiguration. Det kan exempelvis väljas en helt perfekt radiomiljö utan några som helst störningar eller tvärtom, en miljö med stora bakgrundsstörningar som ger upphov till så kallade störningsspikar. Radiomiljön kan även konfigureras med värden för bakgrundsstörningar som mätts upp i lämplig verklig miljö.



Figur 3.3: TOSSIM med tillkopplat GUI (extern applikation) som visar ett tillstånd för en simulation av ett sensornodsprogram.

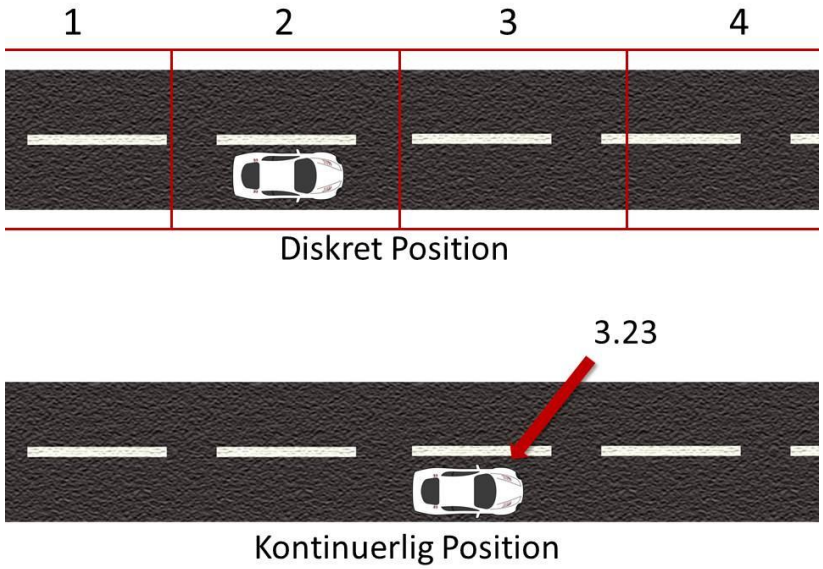
3.4 Trafiksimulatorn SUMO

Det finns många olika typer av simuleringsverktyg för trafik och fordon (Krajzewicz et. al., 2002). Att simulera trafikflöden är mycket svårt på grund av dess höga komplexitet och kaotiska organisation vilket gör att det är svårt att förutsäga kommande händelser. Trafik kan inte enbart beskrivas av bestämda avgångstider och körvägar eftersom människors förflyttning inte är så välplanerad. Hela 60 % av det totala trafikflödet utgörs av dessa mer spontana resor. För denna del av trafikflödet finns inga förutbestämda avgångar och körvägar. På grund av den höga komplexiteten så använder sig simuleringsverktyg av matematiska modeller som liknar verkligheten. SUMO (Simulation of Urban MObility) är ett trafiksimuleringsverktyg. SUMO använder en mikroskopisk vy, se figur 3.4, där fordon eller människor betraktas som de minsta beståndsdelarna i simuleringen (Behrisch et. al., 2011). Trafiksimuleringsverktyget kan användas för att simulera en hel stads trafiknätverk.



Figur 3.4: De olika simuleringsklasserna; makroskopisk modell simulerar trafik som flöden, mikroskopisk modell simulerar enskilda fordon och sub-mikroskopisk modell simulerar ner till enskilda fordonsdelar.

Modellen SUMO använder är kontinuerlig med avseende på plats, se figur 3.5, men inte tid.



Figur 3.5: Skillnad mellan diskret och kontinuerlig position.

En diskret modell använder cellulär automata, vilket innebär att fordonen "hoppar" från cell till cell (Brockfeld E. et al., 2001). Att använda en modell som är diskret och inte kontinuerlig innebär en förenkling som gör att modellen blir mindre verklighetstrogen. Den stora fördelen med diskreta modeller är dock att det blir mindre beräkningar vilket innebär en förenkling som gör programmet mer resurssnålt.

Simuleringen är multimodal vilket innebär att det inte bara är bilar förflyttningar som simuleras utan även alternativa transporthjälpmedel som bussar och tåg. En person har starttid och en resväg som den följer men denna väg kan vara uppdelad i flera delar så att personen åker bil en sträcka, sedan tar bussen o.s.v. Varje fordon som förflyttas inom det simulerade vägnätet simuleras individuellt och har en position och hastighet för varje simuleringssteg. Varje simuleringssteg är 1 sekund lång och efter varje steg uppdateras positionen och hastigheten för varje fordon beroende på andra fordonen i närheten samt den väg fordonet befinner sig på. Modellen är alltså tidsdiskret men platskontinuerlig. Att följa en individuell bil och få information om dess position och hastighet är användbart då trafiksimuleringsverktyget sammankopplas med Gulliver-robotarna, mer om detta i avsnitt 4.1.2.2.

3.4.1 Algoritmer för förarbeteende

Den fordon-förarmodell som används i SUMO kallas Gibbs-modellen (Krauss, 1998). Gibbs-modellen skapar viktiga trafikfenomen som t.ex. köbildning. I varje simuleringssteg är ett fordons hastighet anpassat till det framförvarande fordonet så att ingen kollision kan ske vilket leder till ett kollisionsfritt system. Hastigheten hos ett fordon som inte leder till någon kollision beräknas med hjälp av ekvationen i figur 3.6:

$$v_{aktuell}(t) = v_{framförvarande}(t) + \frac{t_{avstånd} - v_{framförvarande}(t) * t_{reaktion}}{\frac{v}{b_{retardation}(v)} + t_{reaktion}}$$

$v_{aktuell}(t)$ – *Hastighet hos det aktuella fordonet som inte leder till kollision.*

$v_{framförvarande}(t)$ – *Hastighet hos det framförvarande fordonet.*

$t_{avstånd}$ – *Avstånd mellan det aktuella fordonet och det framförvarande i tid.*

$t_{reaktion}$ – *Reaktionstid hos föraren.*

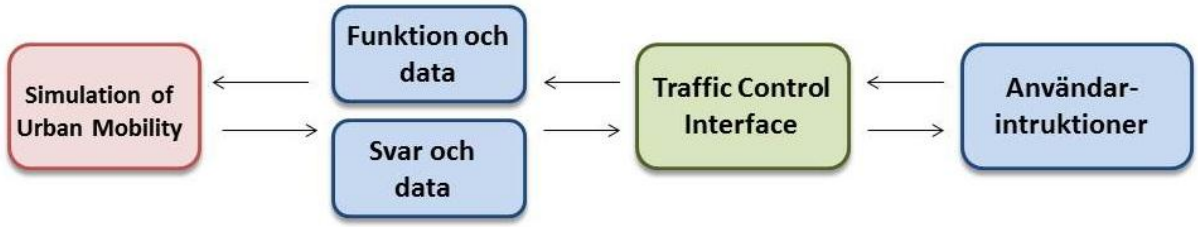
$b_{retardation}(v)$ – *Retardations funktion dvs. hur snabbt fordonet kan bromsa in.*

Figur 3.6: Algoritm för förarbeteende

När hastigheten som inte leder till kollision med framförvarande fordon beräknats för det aktuella fordonet behövs det också kontrolleras så att denna hastighet är fysiskt möjlig för fordonet att uppnå. Därför tas hänsyn till fordonets möjliga maxhastighet och maximala acceleration, för att säkerställa att fordonet inte kör eller accelererar snabbare än vad som har bestämts möjligt för den modell som fordonet tillhör. Förarens förmåga att hålla en bestämd hastighet antas vara imperfekt och därför subtraheras en slumpmässig avvikelse från den optimala hastigheten. Förarens förmåga att hålla den optimala hastigheten kan ändras vid design av ett simuleringsfall.

3.4.2 Traffic Control Interface

TraCI (Traffic Control Interface), är ett gränssnitt som ger åtkomst till pågående simuleringar i SUMO. För att skicka data mellan användarens program och trafiksimuleringsverktyget används TCP. TCP tillhandahåller en dataström mellan två datorer eller en koppling mellan olika program på en dator. En klient/server arkitektur används för att styra simuleringen utifrån vilket kan ses i figur 3.7. SUMO agerar server och svarar via TCP på de förfrågningar som klienten gör via TraCIs kommandon.



Figur 3.7: SUMO agerar server och svarar via TCP på TraCIs kommandon som användarens program har anropat.

För att simulera storskaliga trafikmiljöer behövs ett effektivt simuleringsprogram. Programmet behöver kunna simulera fordonen i detalj, på ett sätt som gör det möjligt att få ut önskad information för varje enskilt fordon. Denna information är t.ex. position och hastighet. Det behöver även vara *open source* eftersom den måste vara lättillgänglig och öppen för vidare utveckling. Systemet kräver också att trafiksimuleringssituationerna är styrbara utifrån den information som simuleringen får från Gulliver-robotarna. Då SUMO tillsammans med TraCI är *open source*, simulerar trafik ned till fordonsnivå och tillåter att trafiksimuleringsscenariorna är styrbara utifrån, är dessa det bästa valet för testsystemet. Dock sågs övriga trafiksimuleringsprogram över men hade bristande dokumentation och saknade den bredd av funktionalitet som projektet behöver.

En nackdel med att programmet är *open source* innebär att det inte finns någon officiell support samt att det inte finns några garantier för att funktionalitet är implementerad och testad. SUMO är inte från början tänkt att interagera med fysiska fordon i trafiksimuleringen, så som projektet kräver. Det gör att en integration till projektets plattform tar längre tid än om SUMO hade varit specialdesignat just för plattformens behov.

3.5 Videoströmning

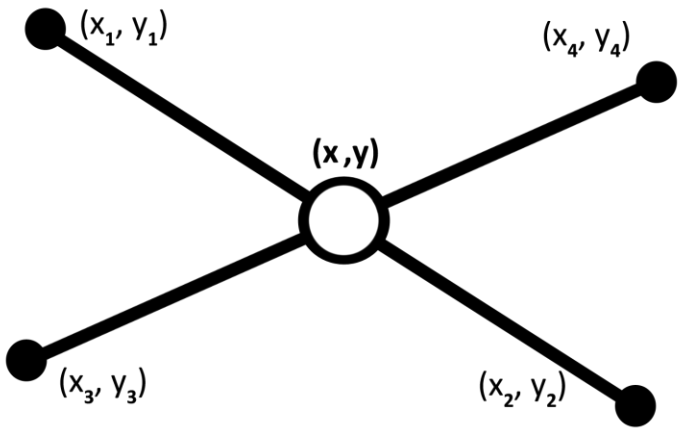
Videoströmning innebär att strömma video från en klient till en annan över ett digitalt nätverk som t.ex. Internet. I detta projekt syftar videoströmning till att i realtid betrakta robotarnas synvinkel genom en webbkamera som är fäst på robotens framsida.

3.6 Generella algoritmer

I detta avsnitt presenteras de generella algoritmerna som används vid verifiering av skapade kartor i kartklienten.

3.6.1 Två linjers skärningspunkt

För att lösa och förenkla vissa delproblem används en matematisk algoritm som räknar ut skärningspunkten mellan två linjer, se figur 3.8. Denna algoritm, den så kallade *line to line intersection* (Akenine-Möller, Haines, Hoffman, 2008), använder sig av förhållandevis enkel matematik som kan ses i figur 3.9.



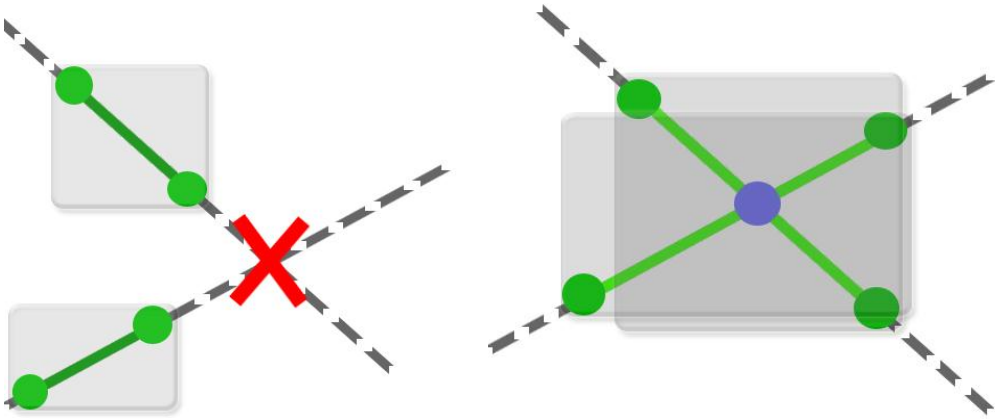
Figur 3.8: Visar skärningspunkten mellan fyra noder.

$$(P_x, P_y) = \left(\frac{(x_1 y_2 - y_1 x_2)(x_3 - x_4) - (x_1 - x_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1 y_2 - y_1 x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \right)$$

Figur 3.9: Uträkningen för skärningspunkten. Om $(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4) = 0$ är linjerna parallella med varandra, annars är punkten (P_x, P_y) skärningspunkten för de båda linjerna.

3.6.2 Grafisk algebra

För att snabba upp och förbättra vissa uträkningar skapas så kallade *bounding boxes*, alltså områden kring två punkter (Akenine-Möller, Haines, Hoffman, 2008). Vid de fall som områdena ej överlappar varandra ignoreras uträkningen av skärningspunkten då denna inte indikerar en "verklig" korsning. Om så är fallet ignoreras denna varning och uträkningarna går vidare, se figur 3.10.



Figur 3.10: Vänster: Områdena kring de två linjerna överlappar ej varandra, därför utlämnas uträkningen av skärningspunkten. Höger: De båda områdena överlappar varandra och kontrollen av skärningspunkt fortskrider. Märk dock att trots områdena överlappar varandra, betyder det inte nödvändigtvis att skärningspunkten ligger mellan de båda linjerna.

3.7 Programmeringsspråk

I plattformen som skapas ingår ett antal olika komponenter som exempelvis trafiksimulatorn SUMO eller Gulliver-robotarnas MICAz-noder. De har alla olika struktur och funktionalitet. Därför använder komponenterna olika programmeringsspråk som lämpar sig för respektive syfte. Nedan följer de olika språk som används av systemets olika komponenter.

3.7.1 XML

XML (eXtensible Markup Language) är ett språk som kan spara strukturerad data på ett sätt som gör det enkelt att flytta data mellan olika program (Evjen et al., 2012). Det är mycket enkelt att använda och är lämpligt för att spara diskret data som används av simulationer och kartor. Trafiksimulatorn SUMO sparar data om simulationer i XML och kartklienten sparar detaljerad information om de skapade kartorna i XML.

3.7.2 NesC

Applikationer som körs i MICAz-nodernas operativsystem TinyOS är skrivna i lågnivåspråket NesC (Gay et al., 2003), vilket i sin tur är baserat på det maskinorienterade språket C. Därför behöver all programmering av TinyOS applikationer ske i NesC. NesC är händelsebaserat och har bra abstraktioner som gör det enklare att arbeta med parallellism. Om en händelse inträffar som kräver tunga beräkningar kan det skapas en intern process som hanterar beräkningarna i bakgrunden utan att blockera nya händelser från att registreras och hanteras.

3.7.3 Python

Simuleringar i nodsimuleringsverktyget TOSSIM skrivs och konfigureras med hjälp av högnivåspråket Python (Leping et al., 2009). Med hjälp av språket byggs simulationer upp som interagerar med den underliggande miljön. SUMO:s gränssnitt TraCI, se avsnitt 3.3.1 och 3.4.2, är också skrivit i Python och möjliggör direkt påverkan av SUMO-simulationer under körning. Detta medförde att det var oundvikligt att använda Python i tesplattformen.

3.7.4 Java

Java är ett objektorienterat språk (Gosling et. al., 2005) som har en mycket stor uppsättning hjälpbibliotek som underlättar exempelvis datahantering och kommunikation. Hjälpbiblioteken har använts bland annat för att behandla information i XML-filer och för att kommunicera med MICAz-noder. Denna funktionalitet i samband med projektmedlemmarnas förkunskaper, vilket undvek en omfattande inlärningsprocess, bidrog till att Java därför blev det naturliga förstahandsvalet vid lösningar där nya komponenter inom plattformen skapades. Ett exempel där Java används är vid implementeringen av kommunikationshanteraren, se avsnitt 4.1.2.2.

3.7.5 C++

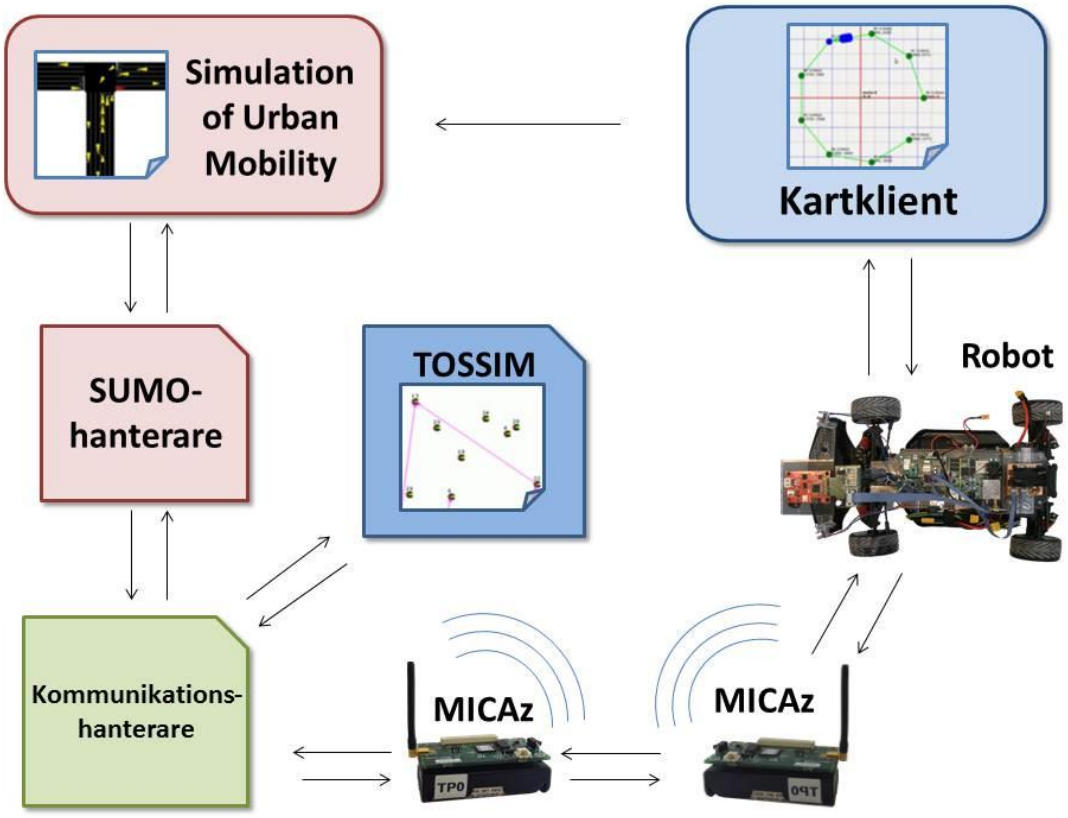
C++ är ett objektorienterat programmeringsspråk (Horstmann, 2012) med många användningsområden. Den skapade kartklienten som används för att definiera kartor för fordonen, bygger på redan existerande mjukvara som är skriven i C++. Denna mjukvara tillåter kommunikation med Gulliver-robotarna genom fjärrstyrning. Kartklienten har därför utvecklats i utvecklingsplattformen Qt, som bland annat stödjer C++.

4 Resultat

I föregående kapitel presenterades tekniker och verktyg som har använts projektet. I detta kapitel beskrivs de lösningar som tillgodoser de mål som satts upp för plattformen. Som en del av kandidatarbetets resultat behövde en mängd nya funktioner implementeras. Dessa funktioner möjliggör genomförande av testfall med integrering av Gulliver-robot och de virtuella fordonen. Implementeringar av de nya funktionerna i testsystemet beskrivs i nästföljande avsnitt 4.1 och därefter beskrivs valideringen av dessa nya implementeringar i testsystemet genom de genomförda testfallen i avsnitt 4.2.

4.1 Implementering av plattformen

För att tillgodose de mål som sattes upp för projektet i avsnitt 1.3, vilket var bland annat att sammankoppla fysiska samt virtuella fordon, har denna systemdesign skapats, se figur 4.1. Denna mångfald av delsystem, som för första gången kopplats samman, har givit möjligheter till att på ett nytt och smidigt sätt utföra komplexa trafiktester till en låg kostnad.



Figur 4.1: Översiktsbild av plattformen.

Plattformen som skapats är en integration av olika specialiserade delsystem. Nedan presenteras den design samt implementering som genomförts för att skapa systemet.

4.1.1 Kartklienten

Med hjälp av Qt har det utvecklats en klient som med ett grafiskt gränssnitt kan köra och testa framtagna teknologier inom intelligenta transportsystem. Klienten är strukturmässigt byggd på ett objektorienterat sätt, där fokus ligger på hanteringen av grafisk interaktion. Denna klient låter användaren lägga till, ta bort och hantera de fordonen som testas i den pågående körningen. Karthanteringsdelen, det fönster där robotarnas rutter kan konstrueras, är implementerad med ett koordinatsystem som med x- och y-koordinater visar robotarnas och vägsegmentens positioner. Denna information fås från robotens positioneringssystem, se avsnitt 3.1. Funktionerna som hanterar kartorna är realiserade med hjälp av knappar som påverkar vilken av de grafiska interaktionsfunktionerna som är aktiv.

4.1.1.1 Utvecklingsverktyget Qt

Projektet har valt att använda utvecklingsplattformen Qt som tillhandahåller funktionalitet för att snabbt och enkelt lägga till olika fönster samt knappar som kan kopplas till specifika egenskaper i koden. Dessa egenskaper underlättar då det inte manuellt behövs justeras i filer för att lägga till knappar och dess koppling.

4.1.1.2 Kartklientens egenskaper

För att skapa de nödvändiga och något komplexa egenskaperna i kartklienten har kunskaperna från framförallt linjär algebra (Sparr, 1994) varit till stor nytta och följande funktioner har lagts till i kartklienten:

Lägga till rutter - med hjälp av denna funktion kan användaren lägga till flera rutter i vägnätet. Vid första utplacerade nod går programmet över till funktionen som lägger till noder på samma rutt istället för att lägga till ytterligare rutter. Positionen av nodens utplacering på kartan räknas ut genom en funktion som använder sig av nuvarande skala och position på kartan samt var muspekaren befinner sig.

Lägga till noder - med hjälp av muspekaren kan användaren enkelt lägga till noder i den aktiva rutten genom att klicka med vänsterknappen på kartan. Utplacering av flera noder kan endast ske om denna funktion är aktiverad.

Ta bort noder - på samma sätt som användaren kan lägga till noder, kan användaren även ta bort noder genom att med höger musknapp klicka inom ett visst närliggande område av noden i den aktiva rutten.

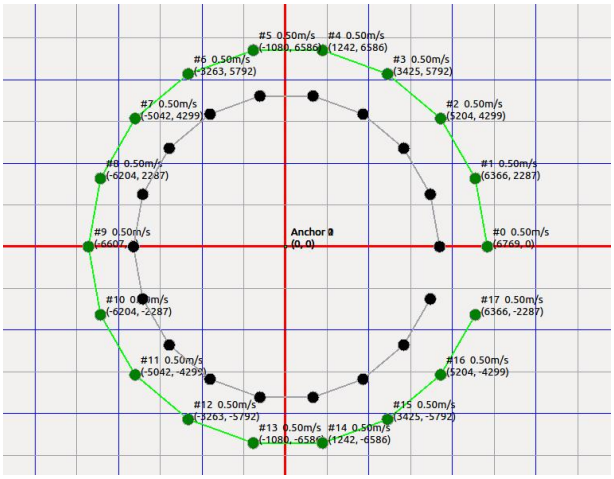
Förflytta enskilda noder - när en användare har placerat ut noder kan denna funktion aktiveras för att med vänsterknappen dra noder till önskat läge.

Förflytta hela rutter - om användaren behöver förflytta hela rutter matar denne in önskad förflyttning i x- och y-led. Dessa värden appliceras sedan på varje enskild nod i den aktiva rutten.

Rotera hela rutter - om användaren behöver finns det möjlighet att rotera den aktiva rutten genom att använda denna funktion. Funktionen tar in ett rotationsvärde som användaren anger som sedan appliceras på den aktiva rutten. Matematiken för detta är att "ta tillbaka" rutten till origo, d.v.s. så att mittpunkten av rutten ligger i origo för att därefter applicera det angivna rotationsvärdet på varje nod och för att därefter "lägga tillbaka" rutten i dess ursprungliga position.

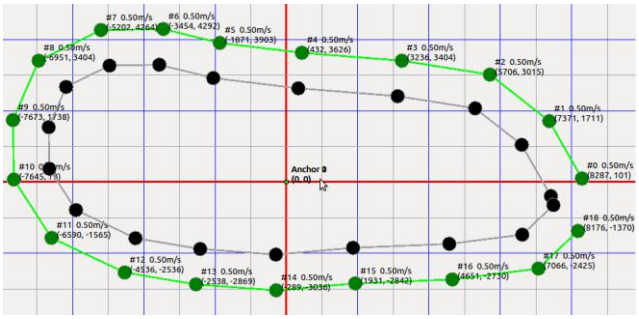
Förstora eller förminska rutter - om användaren vill ha en större eller mindre rutt kan denna funktion förändra den aktiva rutten med ett inmatat värde. Detta betyder alltså att noden förflyttas genom en multiplikation av nodens ursprungliga position med förändringsvärdet. Innan denna multiplikation tar funktionen, på samma sätt som rotationsfunktionen, tillbaka rutten till origo för att sedan applicera förändringsvärdet på varje nod och därefter lägga tillbaka rutten på dess ursprungliga position.

Ändra aktiv rutt - då användaren har flera rutter i sin karta kan denne byta mellan dessa genom att använda denna funktion. Detta är nödvändigt eftersom tidigare nämnda funktioner endast appliceras på den för tillfället aktiva rutten. Det vill säga om användaren vill ändra en nod i en inaktiv rutt, måste denne först använda denna funktion för att stega till önskad rutt. Den aktiva rutten ritas alltid ut med grön färg istället för grå som de inaktiva rutterna målas med, se figur 4.2.



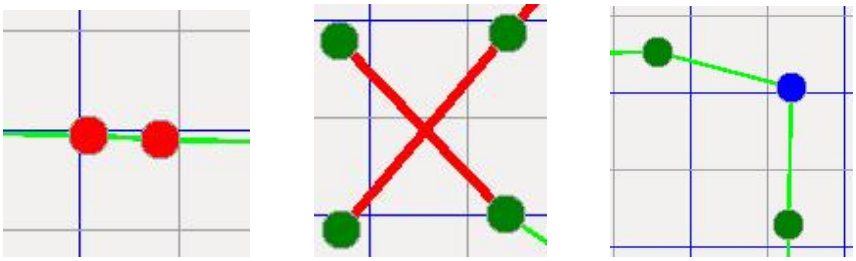
Figur 4.2: Visar en aktiv (grön) och inaktiv (grå) rutt.

Skapa ytterligare fil - användaren har möjligheten att skapa flera filer/rutter genom att använda denna knapp, se figur 4.3. Den nya ruttens noder placeras så att de är vinkelrätta men även på ett fixt avstånd från den aktivas noder. Detta är användbart då komplicerade eller långa rutter kräver två eller flera filer.



Figur 4.3: Användaren har skapat en ytterligare fil av den tidigare aktiva rutten

Validering av rutter - i denna funktion går klienten igenom tre kontroller av alla rutter och deras noder, se figur 4.4. Först kontrolleras varje nod i varje rutt så att noder ej ligger för nära varandra. Därefter kontrolleras om det finns en skärningspunkt mellan nodernas linjer i både samma och övriga rutter. Denna skärningspunkt kontrolleras med hjälp av de tidigare nämnda områdena som undviker ytterligare uträkningar som inte är inom dessa två. Det första området är kring de noder som för tillfället kontrolleras och det andra området är kring de noder som jämförs emot. Slutligen kontrolleras det så att varje efterföljande nod ligger på en sådan position att infallsvinkeln till denna ej överstiger fordonets svängradie. Förutom att användaren meddelas om vilka noder och vilka rutter som behöver justeras, ritas även dessa ut med en annan färg.



Figur 4.4: Vänster: Två noder ligger för nära varandra. Mitten: Två linjer korsar varandra. Höger: Den följande noden ligger i en för snäv kurva sett från ingångspunkten.

Ändra en nods hastighet - varje nod har en position men även en hastighet som fordonet maximalt får köra i, och användaren har med hjälp av denna funktion möjlighet att ändra denna hastighet genom att klicka på önskad nod och sedan skriva in önskad hastighet.

Sätta på/av nod-information - för att förenkla för användaren har det även skapat en knapp som skriver ut varje nods ordning i rutten, position på kartan och maximala hastighet. På detta sätt kan användaren enklare se och förändra de noder som kan ställa till problem eller liknande.

Ta bort aktiv rutt - användaren har möjlighet att med ett enkelt knapptryck ta bort den aktiva rutten från kartan.

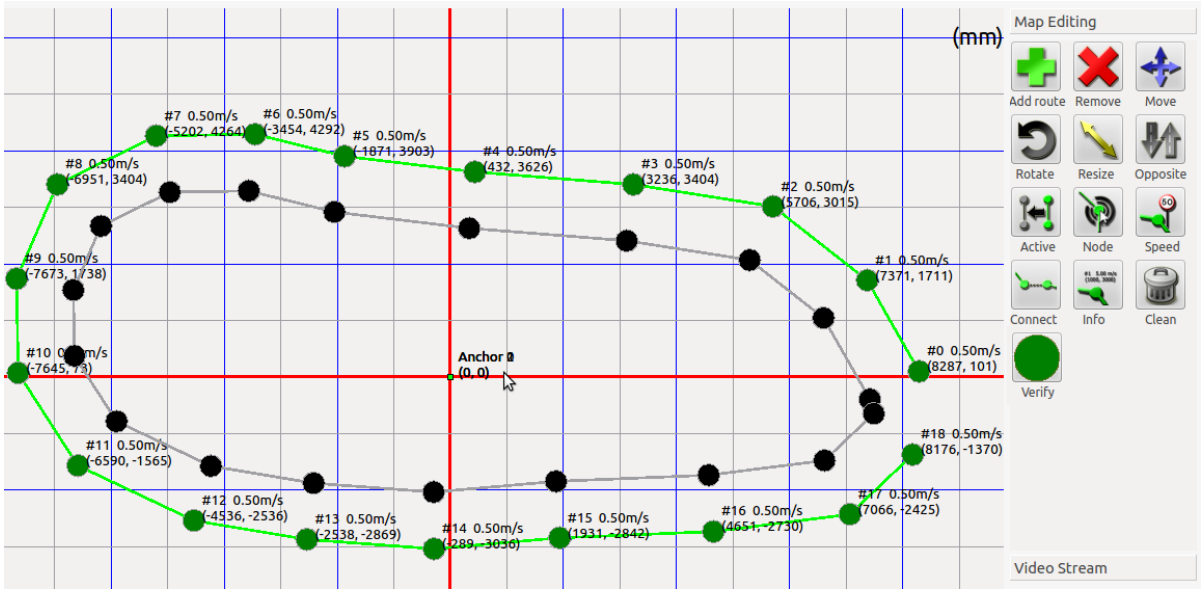
Ta bort alla rutter - om användaren vill börja om från början kan denna funktion ta bort alla rutter som är tillgängliga från kartan.

Skapa areor - användaren har tillgång till att skapa så kallade areor som kan markera eventuella trafikljus eller övriga varningar.

Generera XML-filer till SUMO - Denna funktion finns för att skapa en brygga mellan kartklienten och de virtuella fordonen i trafiksimulatorn. Funktionen genererar XML-filer utav nuvarande vägnät i kartklienten som sedan kan används i SUMO, se avsnitt 4.1.2.1 och 4.1.2.5.

4.1.1.3 Design av grafiskt användargränssnitt

Under utvecklandet av kartklienten sattes det upp ett par viktiga förutsättningar vad gäller designen. Tanken var att skapa ett gränssnitt som med grafiska komponenter var användarvänligt för intressenter inom ämnet. Efter diskussion bestämdes en struktur som under ett "människa-dator-interaktions"-perspektiv (Sharp, Rogers, Preece, 2011) skulle passa bra med den förutbestämda funktionaliteten. Denna struktur är baserad på *drag and drop*-tillämpning och att med muspekaren välja de parametrar och egenskaper som önskas. Gränssnitt av denna typ är väldigt användarvänliga då användaren kan relatera till fysiska handlingar när denne pekar på eller drar och släpper objekt på skärmen (Sharp, Rogers, Preece, 2011). Utplaceringen av en ny rutt, ändringar av vägsegment och andra karttillhörande funktioner är alla hanterbara med grafisk beröring. De knappar som finns i kartklienten som ändrar mellan olika hanteringlägen är utlagda med tillhörande bilder pålagda, se figur 4.5. Dessa bilder har skapats var för sig i bildredigeringsprogrammet Adobe Photoshop (Adobe Creative Team, 2009). Valet av bildredigeringsprogram var enkelt, då vikten av valet lades på förkunskaper för att snabbt och snyggt skapa lämpliga bilder. Med en bild som praktiskt visar funktionen i fråga blir användarupplevelsen bättre, mer tid kan spenderas på produktivitet och inte på eventuell uppsökning av tjänst.



Figur 4.5: Ett utdrag av knapparna och arbetsytan i kartklienten.

4.1.1.4 Implementering av videoström

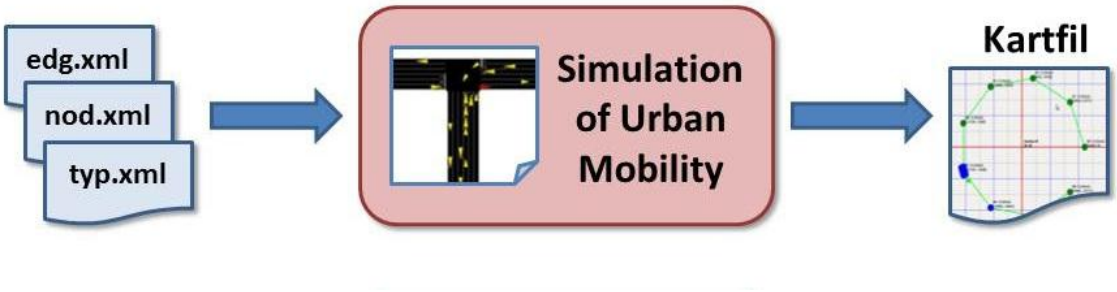
I kartklienten har en funktion för att strömma video direkt från roboten lagts till. Informationen skickas över WiFi och visas med kort fördröjning i ett fönster i klienten. Biblioteket som användes var Phonon (Nokia, 2010) som är skapat för att hantera multimedia i Qt. Uppdateringsfrekvensen för strömmen är ca 15 bilder per sekund, vilket är tillräckligt med tanke på dess uppgift.

4.1.2 Kommunikation och integration

Eftersom trafiksimuleringsverktyget SUMO ska skicka och ta emot information till Gulliver-roboten och TOSSIM så krävs en lösning där de kan skicka information till varandra i realtid. Dessa komponenter behöver även ha en gemensam uppfattning om den miljö de existerar i. Med andra ord kräver SUMO, TOSSIM och Gulliver-roboten en gemensam karta.

4.1.2.1 Trafiksimulatorn

För att skapa ett simuleringsscenario i SUMO krävs en karta som beskriver den miljö de simulerade fordonen framförs i. För att kunna skapa en karta i SUMO behövs information om noder, vilka av dessa som är förbundna med varandra av "vägar" och i vilken riktning dessa vägar går. Vidare behövs information om vilken hastighet är tillåten för specifika vägsträckor m.m. All denna information sparas i tre filer, se figur 4.6, som ligger till grund för den karta som skapas av SUMO.

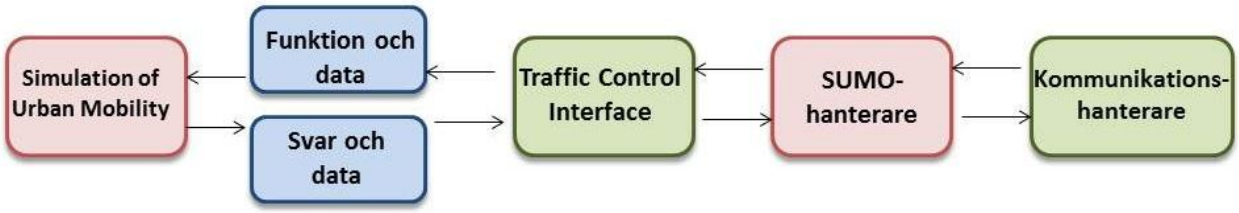


Figur 4.6: Med hjälp av tre informations filer kan SUMO skapa en karta som sedan kan användas för trafiksimuleringar.

När en simulering skapas definierar användaren vilka fordon som denna innehåller och egenskaper som dessa har såsom maxhastighet, acceleration och retardationsförmåga. I simuleringen bestäms de olika fordonens körväg d.v.s. vilka vägar och filer som fordonet kör på.

SUMO-hanteraren styr och kontrollerar simuleringen i SUMO genom att exekvera två parallella trådar. En av dem skriver till kommunikationshanteraren och håller denna uppdaterad med information om simuleringen, se avsnitt 4.1.2. Denna information kan exempelvis vara positionen som ett fordon har, hastighet m.m. Den andra parallella tråden tar in information från kommunikationshanteraren, detta kan exempelvis vara information från andra delar av systemet om hur scenariot har förändrats där. Anledningen till implementeringen av de två parallella trådarna är att kunna utföra kommunikationen i båda riktningar parallellt. SUMO-hanteraren tar även emot och skickar information till SUMO med hjälp av TraCI, som är ett gränssnitt utvecklat för detta ändamål. TraCI packar informationen i paket som skickas till SUMO där informationen

används för att ändra den pågående simuleringen, se figur 4.7. Närmare om hur TraCI sköter denna koppling finns i avsnitt 3.4.2.



Figur 4.7: Visar hur SUMO-hanteraren är sammankopplad med SUMO och kommunikationshanteraren.

4.1.2.2 SUMO till delsystemen

I detta avsnitt beskrivs i detalj hur kommunikationen utförs mellan SUMO och plattformens olika komponenter. Själva "hjärnan" i lösningen är ett Javaprogram, kallat kommunikationshanterare. Här styrs hur, när och vilken information som skickas mellan alla parter. I figur 4.8 illustreras en överskådlig sammanfattning av hur de olika komponenterna kommunicerar med varandra. Kommunikationen mellan SUMO och Java sker via ett Python-program som körs i kommunikationshanteraren. Detta program kallas SUMO-hanteraren och arbetar med parallella trådar för att information skall kunna skickas i båda riktningarna samtidigt. Exakt hur hanteraren interagerar med SUMO förklarades i avsnitt 4.1.2.1.

Själva kommunikationshanteraren startar ovannämnda SUMO-hanterare för att kommunicera med SUMO och initierar en anslutning till en seriell port för att skicka och ta emot information som är till och från Gulliver-roboten. Likt SUMO-hanteraren exekverar även kommunikationshanteraren två parallella trådar för att skicka och ta emot meddelanden samtidigt. Kommunikationshanteraren startar också en anslutning till TOSSIM för att skicka information mellan SUMO och TOSSIM, detta förklaras ingående i avsnitt 4.1.2.4.

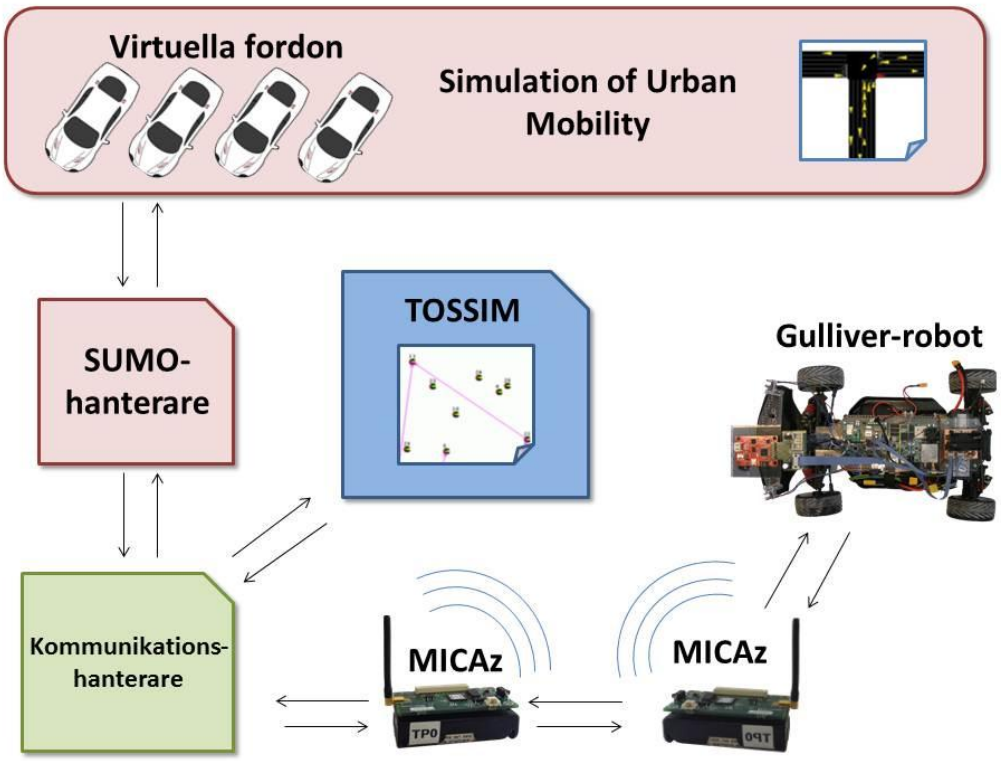
Informationen som skickas från SUMO till kommunikationshanteraren skickas sedan vidare ut till TOSSIM och Gulliver-robotarna. Detta betyder att både fysiska och simulerade fordon har tillgång till informationen som skickas från SUMO. Fordonen kan då välja vilken information de vill behandla baserat på deras implementerade förarlogik. När informationen skall skickas ut till Gulliver-robotarna är en MICAz inkopplad till den seriella porten på datorn. Denna MICAz tar emot information från kommunikationshanteraren och gör om till den paketstruktur som används för att skicka radiomeddelanden via MICAz:or. Här tas också meddelanden som skickas från roboten till SUMO emot via radio och skickas vidare till kommunikationshanteraren via den seriella porten.

Gulliver-robotens MICAz tar emot de meddelanden som den skall behandla, den tar även emot information som är menad för andra fordon då den behöver ha information om t.ex. var närliggande fordon befinner sig. Med denna information kan roboten ta önskvärda beslut och även skicka tillbaka information om sin egen position och fart till SUMO via radion. I avsnitt 4.1.2.3 förklaras hur kommunikationen mellan robotens MICAz och dess *main board* sker. Den information

som är menad till SUMO har definierats med en specifik paketstruktur. Denna paketstruktur innehåller följande:

- Mottagar-id: Varje fordon, virtuellt som fysiskt, har ett unikt id som definieras i SUMO.
- Körfälts-id: Det specifika körfält som fordonet befinner sig i.
- Körfältsposition: Den position i körfältet som fordonet befinner sig på.
- Hastighet: Den aktuella hastigheten för fordonet.

Denna paketstruktur är ett exempel på vad som kan ingå i ett informationsmeddelande. Den är flexibel på så sätt att information kan läggas till eller tas bort i paketstrukturen om så önskas. En sådan ändring i paketstrukturen sker i kommunikationen med Gulliver-roboten och även med TOSSIM vilket förklaras i avsnitt 4.1.2.3 samt 4.1.2.4. Anledningen till att körfälts-id och körfältsposition används istället för absolut position, x- och y-koordinat, är en skillnad i hur de fysiska Gulliver-robotarna och de simulerade fordonen använder sin karta. Robotarna kör enligt sin karta genom att fortlöpande beräkna en möjlig väg till närmaste nod. Detta leder till att de inte kör exakt i kartans körfält på grund av de fysiska begränsningar som finns för en sådan precision. Däremot befinner sig de virtuella fordonen i trafiksimulatorn alltid på en exakt punkt i sina körfält. De har alltså inga fysiska störningsmoment att ta hänsyn till. Lösningen för att integrera dessa olika perspektiv är att frångå att skicka absolut position och istället skicka vilket körfält fordonen befinner sig på. Detta kan ses i paketstrukturen.



Figur 4.8: Visar åt vilka riktningar kommunikation sker och vilket delsystem som pratar med varandra.

4.1.2.3 MICAz till Gulliver-robot

När MICAz väl mottagit meddelanden från kommunikationshanteraren via radio så behandlar den meddelanden enligt sin förarlogik och skickar vidare paket som är anpassade för roboten. Detta sker genom en etablerade tvåvägsanslutning via den seriella porten. Paketstrukturen ämnad för roboten innehåller följande:

- Paket-id: Vilket nummer paketet har i ordningen.
- Hastighet: Vilken hastighet som roboten ska hålla näst
- Paket-räknare: Hur många paket som skickats hittills.

Programmet i roboten tar sedan emot paketet, extraherar informationen om vilken bana och hastighet den ska köra med och följer instruktionen enligt sin logik och körförmåga. Parallellt med detta så skickar även roboten meddelanden till MICAz-noden via seriella porten. Även här skickas ett anpassat paket, men denna gång till noden. Paketstrukturen ämnad för noden innehåller följande:

- Paket-id: Vilket nummer paketet har i ordningen.
- Körfälts-id: Det specifika körfältet som roboten befinner sig i för tillfället.
- Körfälts-framsteg: Hur många procent av det aktuella körfältet som avverkats.
- Hastighet: Vilken hastighet som roboten har för tillfället.

Efter att ha mottagit detta paket så vidarebefordrar MICAz:en meddelandet till kommunikationshanteraren via sin radio.

4.1.2.4 SUMO till TOSSIM

Att ansluta TOSSIM till SUMO sker i två steg. Först initieras simulatorn TOSSIM som körs med hjälp av ett Python-program samt miljökonfigurerande hjälpfiler. Här startas en *serial forwarder* som är ett program i TinyOS verktygslåda. *Serial forwardern* är en TCP-server som ansluter till dem virtuella seriella portarna på sensornoderna i TOSSIM. När miljön är initierad och forwardern är igång skapar kommunikationshanteraren en TCP-anslutning till denna. Detta gör att det nu kan skickas meddelanden till de virtuella fordonen i TOSSIM-simuleringen från SUMO baserat på fordons-id. Samtidigt kan de virtuella fordonen skicka meddelanden till SUMO om de har tagit något beslut som påverkar simulationen. Kommunikationen sker i realtid och paket skickas i den ordning som dem anländer till serial-forwardern. Då TOSSIM är ansvarigt för att hantera radiomiljön behövs ett par extra variabler för att kunna beräkna avstånd mellan fordon och på så sätt ändra radioegenskaperna. Det finns inbyggt stöd för att hämta x- och y-positioner från SUMO som skapar ett tvådimensionellt rutnät för varje simulationsscenario. Detta utnyttjas genom att x- och y-koordinater från SUMO skickas med till TOSSIM där dem kan användas för att ändra radiostyrkan mellan fordon. Nedan beskrivs paketstrukturen för SUMO-TOSSIM kommunikation:

- Fordons-id: Vilket fordon som paketet berör.
- Körfälts-id: Det specifika körfält fordonet befinner sig i.
- Körfälts-position: På vilken position i körfältet befinner sig fordonet.

- Hastighet: Aktuell hastighet för fordonet.
- X-position: Fordonets x-position i ett tänkt två-dimensionellt rutnät, används för att beräkna avstånd till anda fordon och ställa om radioegenskaper.
- Y-position: Fordonets y-position i ett tänkt två-dimensionellt rutnät, används för att beräkna avstånd till anda fordon och ställa om radioegenskaper.
- Meddelandetyp: Tänkt att användas för att urskilja om ett meddelande skall sändas till SUMO eller om meddelandet istället skall sändas via radio till ett fysiskt fordon via kommunikationshanteraren.

Samma paketstruktur används även för radiokommunikation mellan simulerade fordon i TOSSIM.

4.1.2.5 Kartklienten till SUMO och Gulliver-robot

Då de fysiska och virtuella fordonen måste ha en och samma verklighet är det viktigt att alla fordon har överensstämmande karta. Roboten har samma parametrar som kartklienten och det är därför enkelt att ladda upp nuvarande karta till robot. Detta sker med ett knapptryck i klienten. För att kartan skall vara översättbar till SUMO måste det först skapas tre XML-filer, se avsnitt 4.1.2.1, som byggs av att gå igenom alla rutter och spara vad dessa med tillhörande noder. Dessa tilldelas därefter olika namn för att SUMO skall kunna särskilja dem ifrån varandra.

4.1.3 Förarlogik hos Gulliver-robot

För att kunna få Gulliver-robotarnas körstil att efterlikna ett mänskligt förarbeteende har projektet använt trafiksimulatorn SUMO. Genom att plattformen skapar informationsutbyte mellan de fysiska robotarna och SUMO, utökas robotarnas funktionalitet med SUMO:s. SUMO innehåller algoritmer för mänsklig körning, vilka förklaras i avsnitt 3.4.1, och när dessa kan användas av roboten skapas ett människolikt förarbeteende. Roboten innehåller den logik som behövs för att samarbeta med informationen som skickats från SUMO och sända tillbaka information som krävs för testfallet.

4.2 Validering av plattformen

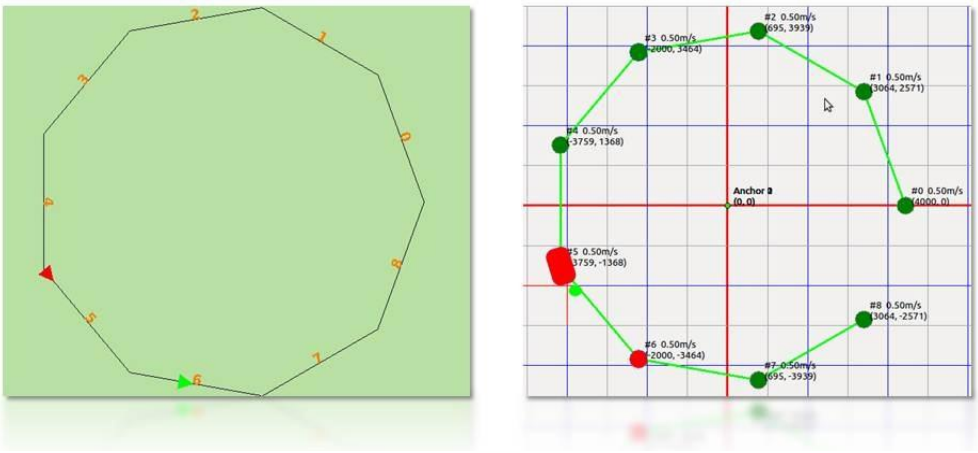
I föregående avsnitt har implementeringen av systemet beskrivits. Med en fullständig integrering av de olika byggda delsystemen skapas den önskade plattformen med den komplexitet som eftertraktats. För att säkerställa och validera att plattformen löser de problem som sattes upp i kapitel 2, har det modellerats två grundläggande testfall som bekräftar funktionalitet, koncept och design.

De två testfallen har valts just för att de visar att plattformen fungerar med de ihopsatta delsystem. En beskrivning av vilken specifik teknisk funktionalitet som de olika delsystemen bidrar med finns i avsnitt 4.1. Då testfallen validerar plattformens mål och funktionalitet, visar de även på att framtida testfall kan genomföras i systemet.

4.2.1 Testfall - Integration av virtuellt och fysiskt fordon

Det första testfallet är en integration av ett virtuellt och en Gulliver-robot. Kartan som båda fordonen kör efter är egenhändigt skapad i kartklienten. Det virtuella ledande fordonet, sett i medurs riktning i figur 4.10, finns bara i SUMO. Information om dess position skickas till den efterföljande roboten som i sin tur anpassar sin position efter detta. Roboten har i uppgift att hålla

ett inställbart minimiavstånd till det ledande virtuella fordonet. Den väljer enligt sin logik att köra ikapp fordonet eller stanna, beroende på om den överskridit minimiavståndet till denna eller inte. Minimiavståndet till denna kan definieras av den som styr simulationen. Roboten skickar information tillbaka till SUMO och kartklienten om dess nuvarande position, denna information används för att uppdatera simuleringsscenariot genom att flytta fordon två till robotens position. På detta sätt möjliggörs realtidsuppdatering av simuleringen.

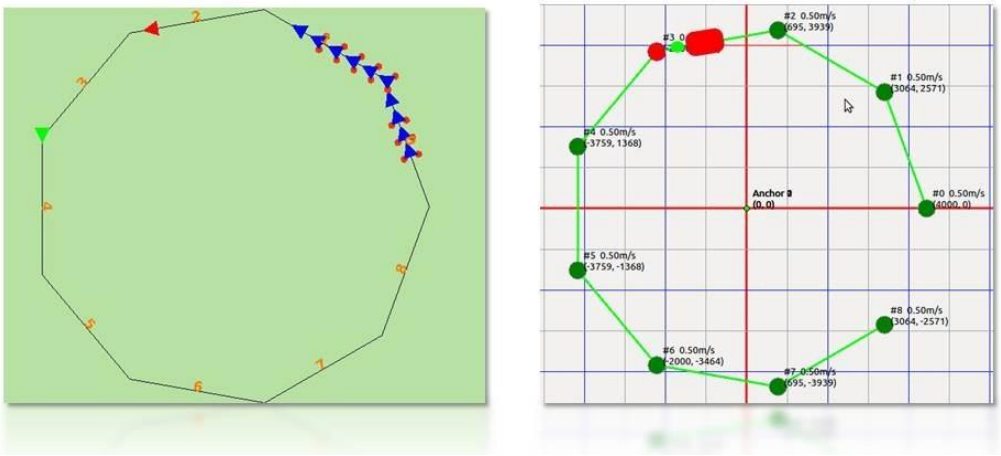


Figur 4.10: Till vänster SUMO:s GUI, till höger kartklienten. Kartklienten visar robotens position som den röda kvadraten. Robotens position överensstämmer med det andra fordonets position.

Slutsatsen som kan dras efter detta testfall är att integreringen av systemet är fungerar och att realtidskommunikationen fyller sin funktion. Då detta testfall har validerats så visar det även på att en exploatering med fler fordon i systemet, både virtuella och fysiska, är möjlig. Eftersom en lyckad integrering har uppnåtts kan även annan information, än den i testfallet ovan valda, skickas. Denna information skulle exempelvis vara acceleration och hastighet för de olika fordonen som skulle möjliggöra en mer komplex logik i roboten, liknande den ekvationen för förarlogik i avsnitt 3.4.1. Detta är dock utanför arbetets område.

4.2.2 Testfall - TOSSIM-simulerade fordon

Detta testfall är en utveckling av det tidigare beskrivna testfallet men här har även en del av fordonen simulerats i TOSSIM. Nedan visar figur 4.11 testfallet. Det gröna fordonet finns bara simulerad i SUMO, medan det röda fordonet är Gulliver-roboten, de blå fordonens position skickas till TOSSIM. I TOSSIM finns de blå fordonen representerade som noder. De har där tillgång till en radiokommunikation som liknar den hos de fysiska fordonen.



Figur 4.11: Vänster: SUMO:s GUI. Höger: Kartklienten visar robotens position som den röda kvadraten.

Det ovan beskrivna testfallet visar på att integration med TOSSIM i plattformen validerats. TOSSIM utökar med en ytterligare dimension av kvantitet på testningen, med möjligheter att simulera flera beslutsfattande fordon se avsnitt 3.3.1 och 4.1.2.4.

4.2.3 Funktionsvalidering

De testfall som är utvecklade, se avsnitt 4.2.1 och 4.2.2, validerar plattformens användbarhet och design. Detta tillåter en granskning av systemets framtagna funktionalitet.

Genom sammansättning av projektets utarbetade delsystem har följande funktionalitet skapats:

I. Tvåvägskommunikation i realtid mellan trafiksimulator och Gulliver-roboten
   - I testfallen valideras realtidskommunikationen genom att observera när SUMO uppdaterar sin position av den fysiska roboten i simuleringen då den tar emot meddelande från Gulliver-roboten. Roboten stannar och kör beroende på avståndet till det virtuella fordonet, den tar konstant emot information från SUMO om var denna befinner sig. Att rätt information mottogs av både SUMO och roboten utan för lång fördröjning, är avgörande för att testfallet skall fungera. Hur denna kommunikation ser ut beskrivs i avsnitt 4.1.2.2.

II. Tvåvägskommunikation i realtid mellan trafiksimulator och nodsimuleringsverktyg
   - Realtidskommunikationen mellan trafiksimulator och nodsimuleringsprogram valideras i testfallen genom att det kan konstateras att information når TOSSIM inom rimlig tid. Genom kontroll av realtidsloggfiler i TOSSIM ses att önskad information mottagits från SUMO. TOSSIM sänder även tillbaka information om fordonens tillstånd och deras beslut. Mer om hur detta har skapats i avsnitt 4.1.2.4.

III. Gulliver-robotens logik
   - Genom att roboten uppför sig som önskat i testfallen visas att den logik som är implementerad i denna är korrekt. I och med att testfallen där fordon följer efter framförliggande fordon visas kan även ett koncept som *platooning*, där flera fordon

autonomt följer ett första "förarstyrt" fordon, vara möjligt att simulera i systemet. Mer om robotens logik i avsnitt 4.1.3.

IV. Definiering av egna kartor och scenarion i trafiksimulatorn
- Utformningen av de valda testfallen har gjorts i SUMO, därigenom påvisas att det är möjligt att skapa egendesignade scenarion. Dessa scenarion är inte begränsade till de ovan beskrivna utan kan innefatta trafikljus, omkörningar m.m. Hur detta gjorts möjligt beskrivs i avsnitt 4.1.2.1.

V. Simulering i realtid av egendefinierade trafiksituationer
- I testfallen simulerar SUMO de trafiksituationer som har valts. Genom observation valideras att dessa testfall sker som önskat eller om justeringar behöver göras.

VI. Simulering av plattformslogik i nodsimuleringsverktyget
- I testfallet TOSSIM-simulerade fordon, se avsnitt 4.2.2, simuleras virtuella fordon i TOSSIM, som tar egna beslut gällande trafiknavigering. Hur denna funktionalitet skapats presenteras i avsnitt 4.1.2.4.

VII. Fjärrstyrning av Gulliver-robot via kartklienten
- Styrningen av roboten valideras då den följer den utsatta kartan i kartklienten. Det kan observeras att beteendet är likadant i klienten och verkligheten då det faktiska rummet bygger på kartan i det virtuella rummet. Hur detta fungerar förklaras i avsnitt 4.1.2.5.

VIII. Fjärrstyrning av Gulliver-robot via trafiksimulatorn
- I testfallen skickades information från SUMO till roboten. Vid observation av roboten kan dess agerande analyseras för att se om roboten agerar enligt de data som skickas. Detta validerar att fjärrstyrningen fungerar korrekt.

IX. En användarvänlig kartklient
- Den skapade karteditorn bär på komplicerad funktionalitet som är implementerad med en lätthanterlig interaktion. Användarupplevelsen valideras genom användning av klienten. Funktionaliteten ger ett system som fungerar väl för testningen. Mer om detta i avsnitt 4.1.1.3.

X. Perspektivutökning med den tänkta förarens synvinkel genom videoström
- Genom analys av videoströmmen i kartklienten vid båda testfallen valideras denna som en tillgång vad gäller både förståelse samt översikt över hur roboten reagerar vid olika simuleringar, se avsnitt 4.1.1.4.

XI. Definiering samt hantering av egna kartor i kartklienten

- Karthanteringen är en stor tillgång vid testningen och är verifierad genom samspelet i testfallen. Kartorna som används i dessa är egendefinierade med attribut såsom körfälts-id, hastigheter m.m. För mer information om detta, se avsnitt 4.1.1.2.

XII. Uppladdning av kartor från kartklienten till Gulliver-robot
- Kartorna som roboten följer är uppladdade över nätverket och kan direkt användas vid testning. Klienten omvandlar sina kartor till XML-filer som SUMO sedan använder, se avsnitt 4.1.2.5. Då kartklientens kartor fungerar hos SUMO, valideras funktionaliteten genom de båda testfallen.

XIII. Överföring av kartor från kartklienten till trafiksimulatorn
- Att kartan som skapats i kartklienten kan importeras till SUMO valideras genom att denna används under testfallen. Hur detta är gjort beskrivs i avsnitt 4.1.2.5.

Primärt så visar testfallen på en integrering av fysiska och virtuella fordon i samma miljö. De genomförda testerna visar att designen av plattformen uppfyller de mål som är ställda på systemet. Med den ovan beskrivna uppsättning av funktioner som tillsammans bildar den färdiga strukturen, finns det en stor potential inom testning av realistiska trafiklösningar. Detta valideras då de två testfallen demonstrerar den lyckade integreringen, och visar att även mer komplexa trafikfenomen kan testas med hjälp av det byggda systemet.

5 Diskussion och reflektion

I föregående kapitel presenterades implementeringen av plattformen som byggts i kandidatarbetet. Plattformens funktionalitet validerades genom två olika testfall. Nu följer en diskussion av den arbetsmetodik som använts inom projektet samt hur vissa val och faktorer påverkat arbetets gång. Avslutningsvis diskuteras plattformen med användningsområden och förslag av möjliga tillägg.

5.1 Utvecklingsprocess

Att dela upp gruppen i två undergrupper, med ansvar för olika delar av plattformen, har fungerat bra och underlättat arbetet. Den oberoende verksamheten har möjliggjort en snabbare utveckling av systemet. Indelningen av individuella ansvarsområden och uppgifter har medvetet gjorts för att projektmedlemmarna skulle bli specialiserade inom varsitt delområde. Detta har underlättat och sparat mycket tid då inte samtliga i projektet behövt skaffa sig djupgående kunskap inom alla delsystem. Mot slutet av projektet lutade mer av arbetet mot integrering av plattformens olika delar, de två områdesgrupperna fick då ett närmare samarbete. Ett samarbete som underlättades betydligt av den individuella kunskapsspecialiseringen.

Projektets mål och syfte har utvecklats och dynamiskt förändrats under arbetets gång vilket har passat väl med den iterativa arbetsmetoden. Vid det tidiga stadiet av projektet ägnades mycket tid åt kunskapsinhämtning innan det slutgiltiga målet växte fram. Det var svårt att få en överblick över vad som redan var implementerat inom forskningsprojektet Gulliver. Eftersom arbetet har inneburit att många komponenter skall integreras, vilka aldrig har sammankopplats på detta sätt tidigare, blev planeringen något komplicerad. Det innebar att uppskattning av tidsåtgång för olika delmoment blev svårare. Även att uppskatta svårighetsgrad och komplexitet av arbetsmomenten var något som var problematiskt i början men blev lättare ju längre arbetet fortskred. När en fördjupad förståelse införskaffats blev det uppenbart att det behövdes en plattform som sammankopplade de redan befintliga Gulliver-robotarna, samt att introducera nya delar.

För kartklienten fanns redan existerande programkod, kod som skötte direkt styrning av Gulliver-roboten. Detta underlättade att förstå det existerande systemets styrkor och brister, vilket i sin tur innebar att en snabbare förståelse över hur kartklienten skulle realiseras införskaffades.

Simulering av fordon kräver ett trafiksimuleringsverktyg. Valet föll på trafiksimulatorn SUMO, och som tidigare nämnt är det ett program med öppen källkod som inte är färdigutvecklat. Detta innebar komplikationer för arbetet då källkoden var ogynnsamt dokumenterad och många funktioner var ofullständigt implementerade. Trots att ett oväntat hinder uppkom, vilket var att gränssnittet till SUMO, TraCI inte var färdigimplementerat eller testat, överkom projektet detta hinder efter en tids felsökning. Därefter kunde programvaran användas fullt ut, lösningen finns dokumenterad i bilaga C och har även givits till utvecklarna av SUMO och TraCI. Dock var SUMO inte avsett av skaparna för att användas i denna plattform, d.v.s. i kombination med fysiska fordon. Därför krävde implementeringen av kommunikationen mellan trafiksimulatorn SUMO och Gulliver-roboten en hel del resurser. Detta innebar att kandidatarbetet blev det första att undersöka, dokumentera och kartlägga en lösning till en sådan miljö.

Vid integreringen av de olika delsystemen löstes det viktiga informationsutbytet som krävs för den önskade funktionaliteten. Detta tog en hel del arbetskraft men var nödvändigt att genomföra. Även att översätta informationen från ett delsystem till ett annat var inte trivialt. Ett exempel på ett sådant dilemma vidrör informationen som skickas mellan trafiksimulatorn SUMO till de fysiska Gulliver-robotarna, vilket beskrivs i avsnitt 4.1.2.2. Arbetet har lyckats hitta en lösning på det nödvändiga informationsutbytet mellan de olika programmeringsspråken samt deras digitala överföring, både trådlöst via radio eller trådburet via den seriella porten. Detta används exempelvis för databehandlingen av informationen där skickande och mottagande källa måste tolka informationen på samma sätt trots olika tekniker.

Projektet har under arbetes gång skapat egna installationsguider för själva mjukvaran i de olika delsystemen på olika operativsystem, på grund av svårigheter som uppstod vid installationen av dem. Då mjukvaran var operativsystemsberoende och behövde användas på både Linux och Windows fanns det inga färdiga automatiserade installeringsmöjligheter. Detta krävde många timmars fördjupning på diverse forum och i bristfälliga guider för att lösa detta problem. Processen utfördes för att kunna använda både SUMO-mjukvaran samt operativsystemet TinyOS. Då det inte existerade någon komplett guide, lämpade det sig därför att skriva egna guider när lösningen väl funnits, se bilaga B.

5.2 Plattformens funktionalitet

Projektets framtagna plattform har skapat en möjlighet att integrera fysiska och virtuella fordon i en och samma simuleringsmiljö. Detta öppnar upp för möjligheter att simulera storskaliga trafiksituationer och miljöer där tusentals fordon kan interagera. Enkel tillgång till sådan testning underlättar för utvecklingen av framtidens trafiklösningar.

Kandidatarbetets implementerade plattform ger många tänkbara testmöjligheter. Tillämpningarna innebär följande tjänster; kartdefiniering och hantering i en specifikt utformad kartklient, tvåvägskommunikation mellan de båda simuleringsverktyg och Gulliver-robotarna, simulering av egendesignade trafiksituationer i realtid både i trafiksimulatorn och i radiomiljön. Nedan förklaras denna funktionalitetens potential med utökade testfall och vidare arbete.

Projektet har möjliggjort att en användare med hjälp av en kartklient kan definiera sina egna kartor på ett enkelt och smidigt sätt. Dessa kartor kan användas för att simulera och verifiera simuleringsmiljöer där fordon, verkliga som datorsimulerade, nu även har möjligheten att kommunicera med varandra. Kommunikationen är en väsentlig del som tidigare inte existerat för slutanvändaren. Plattformen som byggts inom projektet innehåller ett trafiksimuleringsverktyg och en kommunikationsbrygga har byggts mellan denna och de fysiska fordonen. Därigenom kan beslutsfattandet för varje enskilt fordon flyttas från att vara centralstyrt i simuleringsverktyget till att beslutas i varje fordon. Denna beslutslogik har även möjlighet att bli mänsklig då SUMO tillhandahåller algoritmer för mänskligt förarbeteende och kan direkt föras över till varje fordon.

Kandidatarbetets framtagna plattform har stora möjligheter att utökas med ytterligare funktionalitet. En sådan förlängning av systemet vore att implementera det önskade mänskliga förarbeteendet direkt i fordonen. Då trafiksimulatorn SUMO tillhandahåller de algoritmer som

krävs för sådan logik och projektet redan har definierat var i systemet dessa algoritmer implementeras, är detta en möjlig vidareutveckling. Vidare kan den nya kartklienten förfinas samt utökas med diverse funktioner som i nuläget inte är implementerade. Valideringen skulle kunna utökas så att den korrigerar fel, i dagsläget meddelas endast användaren om felaktigheter som existerar i kartan. De areor som nämndes i avsnitt 4.1.1.2 skulle kunna utökas med parametrar som avslöjar vilken sorts trafiksituation roboten befinner sig i, dessa aktiverar ett beslutstagande om vad som skall utföras. Exempel på sådana utökningar är: virtuella trafikljus, hastighetssänkning i en viss kurva och omkörningsområden.

Potentialen är givetvis enorm vad gäller att utöka både den mänskliga förarlogiken och egenskaper hos kartklienten. När utvecklingen av detta arbete fortlöper, öppnar sig fler och fler möjligheter att vidareutveckla själva plattformen med ännu mer funktionalitet.

6 Slutsatser

Resultatet av projektet är en färdigkonstruerad plattform med en egenutvecklad och innovativ design för testning och framtagande av framtida trafiklösningar. Systemet har skapat möjligheter att låta datorsimulerade och fysiska miniatyrfordon interagera med varandra i en och samma simuleringsmiljö. Projektet har även kopplat ihop en kartklient med plattformen för att en användare på ett lätthanterligt sätt skall kunna definiera sina egna testfall. De datorsimulerade fordonen kan simuleras både i trafiksimuleringsverktyget SUMO och i nodsimulatorprogrammet TOSSIM. Plattformen har validerats genom utförlig testning av framtagna testfall. Dessa testfall har validerat att den önskade funktionaliteten av systemet är konsistent.

Kandidatarbetet har visat ett sätt att konstruera en cyberfysisk plattform som möjliggör testning av stora och komplexa system. Arbetet har implementerat ett system som är baserat på fordon och trafikmiljöer men det finns andra områden där liknande lösningar skulle kunna vara till nytta. Plattformen integrerar människoliknande körning i fysiska samt simulerade enheter och kan styras antingen autonomt, semiautonomt eller manuellt. Områden som kan tänkas nyttja funktionalitet liknande projektets, är exempelvis flyg- och marinsektorn. Även i dessa miljöer ingår interaktion mellan mänskligt och autonomt kontrollerade enheter. Flygplatser och hamnar är också extremt komplexa system med många flygplan eller fartyg i rörelse. Säkerhetskraven är rigoröst höga samtidigt som det finns betydande miljöpåverkan om planerade rutter och logistikkedjor inte är välplanerade. Här skulle liknande system, som denna plattform, kunna byggas för att testa och utveckla metoder för effektivisering och högre säkerhet. Från arbetets konstruerade plattform kan en liknande systemdesign appliceras på exempelvis flygtrafik, skillnaden är att de fysiska hindren byts ut och istället för vägar finns det flygrutter.

Den utvecklade cyberfysiska plattformen öppnar upp för många möjligheter vad gäller testning av trafiksituationer. Plattformen kan i det långa loppet ta utvecklingen ett steg närmre framtidens lösning av säkrare och effektivare transportsystem. Då plattformen är skapad för att vara lättillgänglig, prisvärd och effektiv kan studier föras av ett större antal oberoende utvecklare. Detta är nödvändigt för att snabba upp utvecklingen av lösningar på dagens och morgondagens trafikproblem. Minskning av trafikolyckor och utsläpp av miljöskadliga ämnen är ett mål som kan nås just genom mer intelligenta trafikmodeller. Förhoppningsvis så kan dessa kritiska och relevanta problem i modern tid, lättare finna sina lösningar med hjälp av projektets plattform.

7 Referenser

Adobe Creative Team. (2009) *Photoshop CS4*. Stockholm: Pagina Förlags. (Allt i en bok).

Akenine-Möller, T., Haines, E. och Hoffman, N. (2008) *Real-time rendering*. Wellesley, Mass. : A.K. Peters.

Altby et al.(2012) *Robust och noggrant positioneringssystem baserat på avståndsmätningar mellan mobila noder.* Under tryckning. Göteborg: Chalmers tekniska högskola. (kandidatarbete inom Institutionen för Data och Informationsteknik).

Apache (2012), Apache Subversion, http://subversion.apache.org/ (2012-05-14).

Behrisch, M. et. al. (2011), *SUMO - Simulation of Urban MObility: An Overview* In: SIMUL 2011, The Third International Conference on Advances in System Simulation, 2011.

Brockfeld E. et al. (2001). Optimizing Traffic Lights in a Cellular Automaton Model for City Traffic, *Physical Review E,* vol. 64, nr 05, DOI: 10.1103/PhysRewE.64.056132.

Chan, E., Coelingh, E., Robinson, T. (2010) *Operating Platoons On Public Motorways: An Introduction To The SARTRE Platooning Programme*. | 17th World Congress on Intelligent Transport Systems; 25-29 oktober 2010, Busan.

Cockburn, A., Williams., L. (2000). The Costs and Benefits of Pair Programming . Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000). 21-23 juni 2000, Cagliari.

Crossbow (2008), MICAz datasheet, http://www.openautomation.net/uploadsproductos/MICAz_datasheet.pdf (2012-02-07).

Culler, D. et al (2003) TOSSIM: accurate and scalable simulation of entire TinyOS applications.
SenSys '03 Proceedings of the 1st international conference on Embedded networked sensor systems
november 5–7, 2003, New York .

Eurostat (2011), Transport accident statistics
http://epp.eurostat.ec.europa.eu/statistics_explained/index.php/Transport_accident_statistics (2012-05-02) .

Evjen, B. et al. (2007) *Professional XML,* New York, John Wiley and Sons Ltd.

Figur 3.3**, TOSSIM** med tillkopplat GUI (extern applikation) som visar ett tillstånd för en simulation av ett sensornodsprogram, http://www.tinyos.net/tinyos-1.x/doc/tutorial/imgs/tinyviz-screenshot1.gif.

Gay, D. et al. (2003) The nesC language: A holistic approach to networked embedded systems, *PLDI '03 Proceedings of the ACM SIGPLAN;* 2003, New York, s. 1 - 11.

Google (2012), Google docs, https://docs.google.com (2012-05-02).

Gosling, J. et al. (2005) *Java(TM) Language Specification*, 3rd Edition, Java (Addison-Wesley).

Horstmann, C. S. (2012) *C++ for everyone.* Second Edition. New Jersey: Wiley.

Krajzewicz, D. et. al. (2002) SUMO (Simulation of Urban MObility) - an open-source traffic simulation. *Proceedings of the 4th Middle East Symposium on Simulation and Modelling*; september 2002, Sharjah. s. 183-187.

Krauss, S. (1998), *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics*, PhD thesis.

Leping, V. et al. (2009) Python Prevails. *CompSysTech '09 Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing;* 2009, New York, Article No: 87.

Levis, P., Gay, D. (2009) *TinyOS Programming*, Cambridge: Cambridge University Press.

Levis, P. Rusak, T (2010). Physically-based models of low-power wireless links using signal power simulation *Computer Networks*, Volym 54, Nummer 4, pp. 658 – 673

Myhr, A. (2012) Fordons statistik 2010, Statistiska centralbyrån (SCB), Blad nummer 4.

Nationalencyklopedin, (2012) Sensor, http://www.ne.se/lang/sensor, (2012-05-03).

Nokia - Phonon (2010) http://doc.qt.nokia.com/4.6/phonon-overview.html (2012-04-26).

Pahlavan, M., Papatriantafilou, M., Schiller, E. (2011). Gulliver: A Test-bed for Developing, Demonstrating and Prototyping Vehicular Systems. *Proceedings of the 9th ACM International Workshop on Mobility Management & Wireless Access, MOBIWAC 2011*, oktober 31- november 4, 2011, Miami Beach, USA, s. 1-8.

Sharp, H., Rogers, Y. och Preece, J. (2011). *Interaction design : beyond human-computer interaction*. Wiley, 3:e uppl. The Atrium, Southern Gate, Chichester, West Sussex, UK : John Wiley & Sons Ltd.

Sparr, G. (1994) *Linjär algebra*. Lund: Studentlitteratur.

Sutter Herb, (2005) The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software.
Dr. Dob's Journal, vol.30, nr.3, ss.16-22.

Transportstyrelsen (2010), Polisrapporterad olycksstatistik - årsdata,
www.transportstyrelsen.se/sv/Press/Statistik/Vag/ (2012-05-02).

Xia, F. et. al. (2011). Evaluating IEEE 802.15.4 for Cyber-Physical Systems. *EURASIP Journal of Wireless Communications and Networking*, pp. n/a.

Bilaga A - Ansvarsområden

I denna bilaga beskrivs de områden inom projektet som varje enskild individ i gruppen ansvarar över. Eftersom stora delar av projektet har gått åt till integrering av alla områden så har det varit ett frekvent samarbete mellan samtliga medlemmar. Det behöver därför nämnas att alla individer har hjälpt varandra och har endast ett symboliskt huvudansvarsområde.

A.1 Ansvarsområden

Projektgruppen har huvudsakligen delats upp i två mindre grupper. Den första gruppen har fokuserat på att koppla samman delsystemen i plattformen. Personerna som ingick i denna grupp var Anmar, Erik, Johan och Nadia. Följande lista visar respektive persons huvudansvar:

Trafiksimulatorn SUMO	Nadia
Tvåvägskommunikation i realtid	Johan
Radiosimuleringsverktyget TOSSIM	Erik
Kommunikation med Gulliver-robot	Anmar
Scenarion	Nadia, Johan, Erik, Anmar
Integration med kartklient	Nadia

Den andra gruppen har utvecklat tjänsten för karthantering och bestod av Daniel och Viktor. Deras ansvarfördelning ser ut enligt följande:

Algebraiska implementationer av funktioner	Viktor
Grafisk interaktion	Daniel
Design	Daniel
Videoström	Daniel, Viktor
Integration med trafiksimulator	Viktor

A.2 Framtagning av slutrapport

Vid arbetet av slutrapporten har samtliga medlemmar varit högst involverade. Under planeringen för rapporten har individer fått huvudsakliga ansvarsområden för att dela upp arbetet på ett effektivt sätt. Denna uppdelning ändrades ständigt då olika individer samarbetat inom utvecklingen av olika områden vilket framgår enligt avsnitt A.1. Samtidigt har varje individ korrigerat och tillfört substans inom de flesta delarna av rapporten. Detta innebär att det blir svårt att urskilja vem som har tillfört vad inom projektgruppen och enskilda ansvarsområden för de olika avsnitten är därför strukna.

A.3 Presentationer och opponering

Muntlig mittredovisning	Daniel, Nadia
Demonstration	Samtliga
Muntlig slutredovisning	Anmar, Erik, Johan
Muntlig opponering	Nadia, Viktor

Bilaga B - Installation av TinyOS

I denna bilaga beskrivs hur en komplett installation av TinyOS (beskrivning av TinyOS finns i avsnitt 3.3) går till. Systemet är beroende av TinyOS, men det finns i dagsläget inga bra guider tillgängliga som på ett enkelt sätt beskriver hur installation går till i ett nyare Ubuntu-system. Nedan följer en utförlig guide för hur installationen kan göras samt grundläggande förklaringar till varje moment.

Installera TinyOS 2.1.0 under Ubuntu 11.10

Förutsättningar: PC med installation av Ubuntu Linux ver. 11.10/11.04 med svensk språkuppsättning samt grundläggande färdigheter i Linux.

1. TinyOS är beroende av en fungerande Javamiljö, Bland annat för att skicka paket via serie-/USB-porten och kommunicera med noder. Installera Java (om du inte redan har det installerat) genom att starta en terminal och skriv:
sudo apt-get install openjdk-6-jre

2. För att kunna installera TinyOS måste vi lägga till ett paketförråd så att pakethanteraren kan hitta dem nödvändiga filerna. Detta gör vi genom att lägga till Stanfords TinyOS-paketförråd i sources.list.
Starta *Uppdateringshanteraren*, välj menyn för Inställningar->Övrig programvara->Lägg till
skriv i rutan som kommer upp:
deb http://tinyos.stanford.edu/tinyos/dists/ubuntu lucid main

Även om Lucid inte är den Ubuntu-version som vi kör så använder vi dess paketförråd, det finns många fler och nyare på Stanfords server men flera av dem innehåller korrupta eller ej fungerande uppsättningar av verktygen.

3. För att läsa om alla paketförråd och få tillgång till dem nya paketen, skriv följande i en terminal:
sudo apt-get update

4. Nu är det dags att installera TinyOS. Detta gör vi enklast från en terminal då det blir rätt plottrigt att välja paket i Synaptic. Skriv följande i en terminal för att starta installationen:
sudo apt-get install tinyos-2.1.0

Acceptera alla frågor som kommer upp under installationen.

5. TinyOS är beroende av att ha en fungerande utvecklingsmiljö för Python installerat. Följande kommando installerar Python:
sudo-apt-get install python-dev

6. Dem nya versionerna av Ubuntu använder nyare versioner av Python än version 2.5 som TinyOS är förkonfigurerat med. För att ta reda på vilken version som ditt system använder kan du skriv följande kommando i en terminal:
python --version

7. Därefter skriver vi följande kommando för att starta editering av en konfigurationsfil tillhörande TinyOS för att konfigurera denna att använda vår version av Python:
gedit /opt/tinyos-2.1.0/support/make/sim.extra

Det finns risk att filen är skrivskyddad då vi installerade som administratörer(sudo)
för att enkelt lösa detta kan man skriva följande:
sudo gedit /opt/tinyos-2.1.0/support/make/sim.extra

Nu när vi har behörighet, leta upp raden med *"python_version"* och ändra så att den matchar resultatet från punkt 6. Tänk på att bara använda en decimal så att exempelvis Python 2.7.2+ blir 2.7.

8. Vi har nu installerat, Python, Java och TinyOS samt kopplat ihop Python och TinyOS att fungera tillsammans. Nu kvarstår det att koppla ihop TinyOS med Javainstallationen på datorn. För att installera TinyOS Javaverktyg i din dators Javainstallation skriv:
sudo tos-install-jni

9. Troligen behöver CLASSPATH i *"tinyos.sh"* ändras då denna installeras med en felaktig konfiguration. Skriv följande i terminalen för att starta editeringen av filen:
gedit /opt/tinyos-2.1.0/tinyos.sh

Se till att CLASSPATH-raden ser ut som nedan:
CLASSPATH=$CLASSPATH:$TOSROOT/support/sdk/java/tinyos.jar:.

OBS! Det är mycket viktigt att kolonet och punkten i slutet är med!

10. Vi är nu i princip klara med installationen. För att alla inställningar skall läsas in varje gång vi startar en terminal måste vi lägga till en sökväg till TinyOS konfigurationsfil i systemets bash fil. Skriv följande:
gedit ~/.bashrc

Klistra därefter in följande rad längst ned i dokumentet:
source /opt/tinyos-2.1.0/tinyos.sh

11. Vi är nu redo att testa kompilera vår första applikation. Med TinyOS följer en stor mängd exempelapplikationer automatiskt, den enklaste av dessa är *"Blink"* som helt enkelt med hjälp av en

timer får en LED-lampa att blinka på en nod. För att testa att miljön är korrekt installerad behöver vi dock inte ansluta en fysisk nod, det räcker endast att kompilera applikationen. Skriv följande:
cd /opt/tinyos-2.1.0/apps/Blink
make micaz

(11.5) Om du får fel i stil med att du inte har tillåtelse eller *privilegies* så beror det på att vi installerade TinyOS som administratörer(sudo). För att fixa detta ändrar vi behörighet på mappstrukturen för TinyOS, skriv:
cd /opt
sudo chmod -R 777 TinyOS-2.1.0/

Detta sätter fullständig behörighet för alla användare på TinyOS mappstruktur. En annan behörighetsgrad som bara ger din användare rättigheter funkar naturligtvis också bra.
Gå därefter tillbaka till steg 11 och testa igen.

12. För att testa att kompilera för TOSSIM kör vi nästan samma kommando men med tillägget *"sim"*:
make micaz sim

(12.5) Om detta inte fungerar beror det sannolikt på att Python inte är fungerar korrekt, lägg i sådana fall till raden:
PYTHONPATH=$PYTHONPATH:$TOSROOT/support/sdk/python tinyos.sh
(steg 9) Spara i dokumentet, stäng ned och starta om terminalen. Testa därefter steg 12 igen.

13. Det sista vi behöver kontrollera är om Java-toolchain fungerar som den skall. Detta gör vi genom att kompilera en applikation som har stöd för att via USB/serieporten prata med ett Javaprogram. Skriv följande:
cd /opt/tinyos-2.1.0/apps/tests/TestSerial
make micaz

Om allt fungerar som det skall har du nu en fungerande installation av TinyOS!

Bilaga C - Lösning på problem i SUMO

För att kunna påverka trafikscenarion i SUMO under pågående simulering kan TraCI användas, se avsnitt 3.4.2 och 4.1.2.1. Under projektets gång hittades en bugg när funktionen moveTo användes. Funktionen moveTo flyttar ett fordon till en ny position.

Lösning:
Gå in i SUMO->tools->traci->vehicle.py

På rad 476 i funktionen moveTo, byt ut den första ettan till en tvåa, se figur C.1. Det är paketets längd som definieras här och den är felberäknad. Om man inte ändrar detta så uppstår ett fel när funktionen moveTo används, som säger att mer data var skickat än som var läst.

```
475 def moveTo(vehID, laneID, pos):
476         traci._beginMessage(tc.CMD_SET_VEHICLE_VARIABLE, tc.VAR_MOVE_TO, vehID, 2+4+1+4+len(laneID)+8)
477         traci._message.string += struct.pack("!Bi", tc.TYPE_COMPOUND, 2)
478         traci._message.string += struct.pack("!Bi", tc.TYPE_STRING, len(laneID)) + laneID
479         traci._message.string += struct.pack("!Bd", tc.TYPE_DOUBLE, pos)
480          traci._sendExact()
```

Figur C.1: Utdrag ur koden för vehicle.py i SUMO.

Bilaga D - Vidarebefordrad kunskap

I kandidatarbetet har det varit en viktig del att samarbeta med forskningsprojektet Gulliver och att införliva vårt projekts arbete till detta. För att möjligöra ett sådant samarbete har kandidatgruppen haft regelbundna veckomöten där kunskap och information har utbytts med forskningsprojektets medlemmar. Utöver dessa möten har gruppen även förklarat och visat hur kandidatarbetets nyskapade system fungerar. Genom systemet har gruppen positivt bidragit till och påverkat forskningsprojektets framtagning. Detta i bemärkelse att gruppen har undersökt och införskaffat kunskap samt hittat flertal tekniska lösningar på existerande samt nyfunna problem. Detta har underlättat för projektets mål.

Vid de regelbundna mötena har bland annat Mustafa Mohammed, Mitra Pahlavan, Elad Schiller och Benjamin Vedder varit närvarande. I kandidatarbetes slutfas har det ägnats mycket tid åt att överlämna den konstruerade plattformen. Det har skett genom täta möten, dokumentation och praktiska sessioner. Mustafa Mohammed har följt med i det dagliga arbete under den sista läsperioden för att införskaffa kunskap om alla detaljer i hur systemet är konstruerat och hur det används. Detta för att plattformen skall kunna vidareutvecklas och vara till nytta för forskningsprojektets fortsatta arbeten. Vidare har informella instruktionsvideor skapats som demonstrerar hur systemet sätts upp och används.